Les microcontrôleurs PIC

Bernard Béghyn

Hermes

Lavoisier

LAVOISIER
11, rue Lavoisier
75008 Paris

Serveur web : www.hermes-science.com

ISBN 2-7462-0764-8

Les microcontrôleurs PIC

Bernard Béghyn



Table des matières

Avant-propos

Fabriqués et commercialisés depuis de nombreuses années par la société MICROCHIPS, les microcontrôleurs PIC sont reconnus pour leur rapidité et pour leur facilité de programmation in situ. La série FLASH permet entre autres de nombreux essais et reprogrammations lors de la phase d'étude. Ils sont appréciés autant par les industriels que par les bricoleurs amateurs d'électronique.

Leur architecture permet des opérations rapides, les instructions sont peu nombreuses et leur interfaçage est complet : ports parallèles, convertisseurs analogiques-numériques, temporisateurs, et liaisons série synchrone et asynchrone.

Ils sont déclinés en de très nombreuses versions et chaque application trouve un modèle adéquat.

Choisi par les enseignants de certaines académies comme base dans la réalisation des projets d'étudiants techniciens et techniciens supérieurs, le PIC méritait un abord plus facile, car son constructeur ne fournit qu'une documentation technique destinée aux professionnels et rédigée uniquement en anglais.

Ce livre est destiné à tous ceux qui veulent découvrir les nombreuses possibilités des micro-contrôleurs PIC pour ensuite les utiliser dans une production professionnelle ou encore dans une application domestique. Il contient une étude complète du 16F84 afin de se familiariser avec l'architecture PIC et sa programmation, une étude succinte des interfaces du 16F877, une étude plus approfondie des interfaces série synchrone (SPI et I2C) de plus en plus employées dans l'interfaçage électronique et une prise en main du simulateur du logiciel de développement MPLAB.

Le technicien trouvera dans cet ouvrage les informations nécessaires à une mise en œuvre classique de ce microcontrôleur, toutefois, la mise en œuvre de systèmes professionnels nécessitera l'étude de la documentation fournie par le fabricant.

Cet ouvrage est aussi rédigé à l'attention des étudiants ; d'une part à l'étudiant qui a déjà de solides bases de traitement numérique de l'information ou de programmation et qui peut alors d'emblée utiliser les interfaces et, d'autre part, à

l'étudiant qui commencera par découvrir la structure et le fonctionnement d'une unité centrale d'un microcontrôleur avant de la mettre en œuvre.

Seront intéressés :

- les techniciens et ingénieurs des entreprises ;
- les étudiants des facultés des sciences et des grandes écoles d'électronique et d'électro-technique ;
- les étudiants des IUT et BTS génie électrique et génie électronique ;
- les élèves des lycées techniques et des lycées professionnels.

Bernard BÉGHYN

Chapitre 1
Conception de l'unité centrale

L'ouvrage est destiné à ceux qui ont le projet d'utiliser un microcontrôleur de type PIC de la marque MICROCHIP. Toutefois la découverte de ses possibilités nombreuses exige un certain nombre de prérequis indispensables :

– connaître parfaitement :
  - le codage binaire ;
  - le codage hexadécimal ;
  - le codage BCD ;
  - le codage ASCII ;

– savoir transcoder (passer d'un code à l'autre) ;

– connaître le codage en complément à deux ;

– connaître tous les circuits logiques de base (multiplexeurs, démultiplexeurs, compteurs, etc.), les technologies CMOS et TTL ;

– connaître les mémoires et leur utilisation ;

– avoir des notions d'écriture de logigramme.

Nous commencerons par faire quelques rappels sur des notions indispensables.

1.1. Rappels

1.1.1. *Notion de bus*

Un bus est un ensemble de fils de liaison entre deux circuits logiques, qui permet le transfert en parallèle de mots binaires.

EXEMPLE.– Un bus permettant la transmission en parallèle d'un octet comprendra 8 fils et un fil de masse.

Un bus peut être unidirectionnel ou bidirectionnel, grâce à la logique trois états.

1.1.2. *La logique trois états*

En logique binaire, on utilise deux niveaux actifs de courant ou de tension : le niveau logique 0 et le niveau logique 1.

La logique trois états comporte un état supplémentaire appelé haute-impédance (HZ). Dans ce cas, le circuit se comporte comme si il était déconnecté de la liaison. Sa sortie peut être simulée par un interrupteur ouvert (voir figure 1.1).

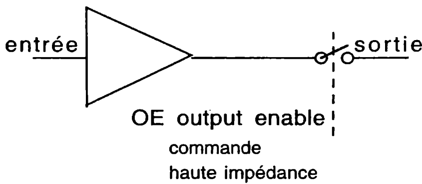
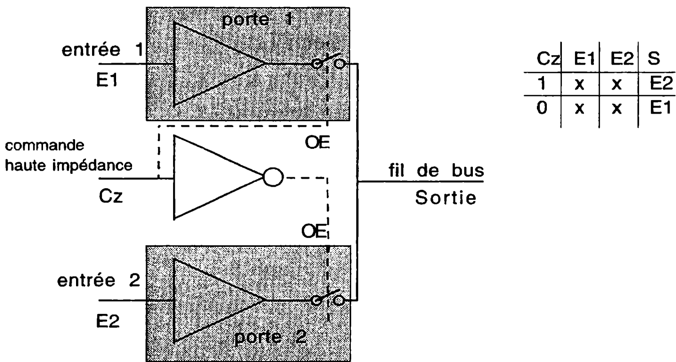


Figure 1.1. *La logique 3 états*

En fonctionnement « normal », un état logique à l'entrée se retrouve en sortie (interrupteur fermé). L'état haute impédance est commandé par une entrée supplémentaire nommée OE (*Output Enable*). Sur la figure 1.1, celle-ci ouvre l'interrupteur avec un niveau logique 1. Ceci permet de brancher plusieurs sorties de circuits logiques sur un même fil de liaison (voir figure 1.2). L'inversion créée au niveau de la commande OE des deux portes, permet un fonctionnement « normal » d'une porte alors que l'autre est en état haute impédance et inversement.



Cz	E1	E2	S
1	x	x	E2
0	x	x	E1

Figure 1.2. *Buffer trois états*

1.1.3. *Application*

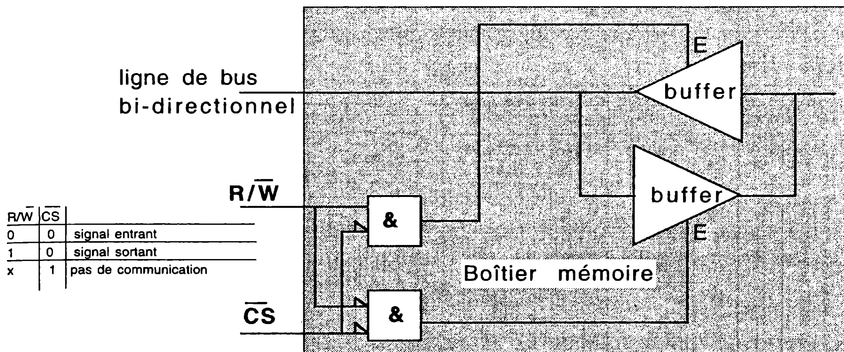


Figure 1.3. *Accès à un boîtier mémoire*

Retrouver le sens de transfert des informations sur la ligne du bus bidirectionnel et l'état de cette ligne en fonction des niveaux logiques sur les lignes CS et R/W (voir figure 1.3). Le signal E permet le fonctionnement « normal » du buffer. La ligne CS permet de valider le fonctionnement de l'ensemble du buffer trois états, un niveau logique 0 est nécessaire pour autoriser le transfert de données dans les deux sens du bus. La ligne R/W permet la sortie de donnée du boîtier mémoire quand elle est à 1 et l'entrée de donnée quand elle est à 0.

REMARQUE.– On retrouvera ces deux signaux dans la suite du cours :

CS (*Chip Select*) veut dire sélection de boîtier.

R/W (*Read/Write*) est le signal de lecture/écriture dans un circuit mémoire :
- le signal est actif au niveau 1 pour la lecture ;
- le signal est actif au niveau 0 pour l'écriture.

1.1.4. *Registre tampon ou accumulateur*

Le terme « accumulateur » désigne la mémorisation temporaire des informations binaires. Il est composé d'un nombre de bascules bistables correspondant à la longueur du mot à mémoriser. Il est capable d'effectuer un certain nombre d'opérations telles que décalage à droite ou à gauche, rotation à droite ou à gauche grâce à un bit supplémentaire, etc.

4-bit arithmetic logic unit — 74HC/HCT181

FUNCTION TABLES

MODE SELECT INPUTS				ACTIVE HIGH INPUTS AND OUTPUTS	
S_3	S_2	S_1	S_0	LOGIC (M=H)	ARITHMETIC[2] (M=L; C_n=H)
L	L	L	L	$\overline{A}$	A
L	L	L	H	$\overline{A + B}$	A + B
L	L	H	L	$\overline{A}B$	$A + \overline{B}$
L	L	H	H	logical 0	minus 1
L	H	L	L	$\overline{AB}$	A plus $A\overline{B}$
L	H	L	H	$\overline{B}$	(A + B) plus $A\overline{B}$
L	H	H	L	A ⊕ B	A minus B minus 1
L	H	H	H	$A\overline{B}$	$A\overline{B}$ minus 1
H	L	L	L	$\overline{A} + B$	A plus AB
H	L	L	H	$\overline{A \oplus B}$	A plus B
H	L	H	L	B	$(A + \overline{B})$ plus AB
H	L	H	H	AB	AB minus 1
H	H	L	L	logical 1	A plus A[1]
H	H	L	H	$A + \overline{B}$	(A + B) plus A
H	H	H	L	A + B	$(A + \overline{B})$ plus A
H	H	H	H	A	A minus 1

Notes to the function tables

1. Each bit is shifted to the next more significant position.
2. Arithmetic operations expressed in 2s complement notation.

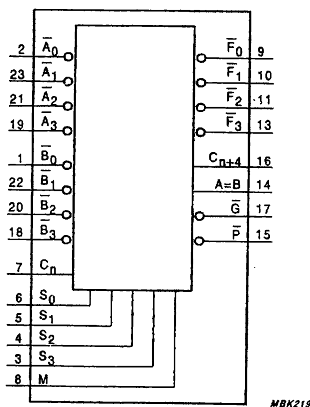
H = HIGH voltage level
L = LOW voltage level

MODE SELECT INPUTS				ACTIVE LOW INPUTS AND OUTPUTS	
S_3	S_2	S_1	S_0	LOGIC (M=H)	ARITHMETIC[2] (M=L; C_n=L)
L	L	L	L	$\overline{A}$	A minus 1
L	L	L	H	$\overline{AB}$	AB minus 1
L	L	H	L	$\overline{A} + B$	$A\overline{B}$ minus 1
L	L	H	H	logical 1	minus 1
L	H	L	L	$\overline{A + B}$	A plus $(A + \overline{B})$
L	H	L	H	$\overline{B}$	AB plus $(A + \overline{B})$
L	H	H	L	$\overline{A \oplus B}$	A minus B minus 1
L	H	H	H	$A + \overline{B}$	$A + \overline{B}$
H	L	L	L	$\overline{A}B$	A plus (A + B)
H	L	L	H	A ⊕ B	A plus B
H	L	H	L	B	$A\overline{B}$ plus (A + B)
H	L	H	H	A + B	A + B
H	H	L	L	logical 0	A plus A[1]
H	H	L	H	$A\overline{B}$	AB plus A
H	H	H	L	AB	$A\overline{B}$ plus A
H	H	H	H	A	A

Notes to the function tables

1. Each bit is shifted to the next more significant position.
2. Arithmetic operations expressed in 2s complement notation.

H = HIGH voltage level
L = LOW voltage level



Les fonctions réalisées par l'UAL 74HC181

1.1.5. *L'unité arithmétique et logique : UAL*

Une unité arithmétique et logique est un circuit combinatoire capable de réaliser des opérations arithmétiques (exemple : addition) ou des opérations logiques (exemple : ou exclusif) sur des mots binaires appliqués sur ses entrées. Ces mots binaires sont appelés opérandes.

Combinatoire veut dire que le résultat est obtenu quasiment instantanément à la sortie du circuit dès la mise en place des données sur les entrées. Le résultat est celui d'une simple combinaison binaire.

L'UAL 74HC181 permet d'effectuer des opérations sur des quartets (mots de 4 bits). La documentation fournie permet de se rendre compte des opérations réalisées par une unité arithmétique et logique.

Elle peut réaliser 16 opérations logiques et 32 opérations arithmétiques selon l'état de l'entrée Cn (16 pour Cn = 0 et 16 pour Cn = 1). Pour sélectionner une opération à effectuer, elle dispose de quatre entrées de sélection de fonctions sur lesquelles on place un mot de commande. Elle dispose aussi d'une sortie qui passe à 1 lorsque les deux mots d'entrée sont égaux :

A = B => fonction comparaison.

Quand Cn = 1 Cn + 4 passe à 1 si A < ou = à B (inférieur ou égal), Cn + 4 passe à 0 si A > B

Quand Cn = 0 Cn + 4 passe à 1 si A < B, Cn + 4 passe à 0 si A > ou = à B, Cn + 4 permet ainsi de mettre plusieurs UAL en cascade.

L'entrée M permet de sélectionner les fonctions arithmétiques ou les fonctions logiques.

Les soustractions sont effectuées sur des nombres en complément à 2.

1.2. Introduction à l'unité centrale

L'unité arithmétique et logique est un circuit combinatoire, or le mode combinatoire ne présente qu'un intérêt limité, il est intéressant de faire fonctionner ce circuit en mode séquentiel.

1.2.1. *Fonctionnement (voir figure 1.4)*

Pour bien comprendre le fonctionnement nous allons essayer de résoudre un problème simple : additionner deux octets X et Y.

Pour réaliser l'addition, on commencera par placer le mot de commande ADDITION sur les connexions de l'UAL qui déterminent les opérations. Ensuite on générera trois opérations (mode séquentiel) :

– opération 1 : on ouvre le chemin D (bus D) et on ferme grâce à la logique trois états les bus A, B et C. L'octet X arrive par le bus de données et va se placer dans le registre accumulateur ;

– opération 2 : on ouvre les chemins (bus) C, B et A, le bus D étant fermé, le deuxième octet Y arrive par le bus de données sur la deuxième entrée de l'UAL, le résultat de l'addition est donc disponible instantanément à la sortie. Le bus B étant ouvert, le résultat vient se placer dans le registre accumulateur et écrase la donnée qui était présente précédemment ;

– opération 3 : le registre accumulateur contient maintenant le résultat de l'addition X + Y = R. Si on ouvre le bus D en fermant les autres chemins le résultat de l'opération peut passer sur le bus de données.

On peut aussi imaginer que le résultat soit rangé dans un autre registre.

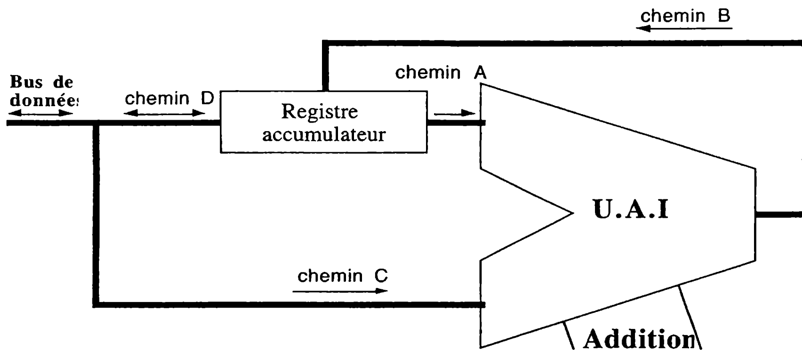


Figure 1.4. *Principe de base*

Les différentes opérations décrites précédemment sont en réalité générées par un automate séquenceur. Celui-ci est piloté par une horloge (mode séquentiel). L'automate est un décodeur qui génère une suite de micro-instructions en fonction du mot binaire placé dans un registre appelé registre d'instructions.

Ces micro-instructions sont entre autres dans notre exemple les commandes d'ouverture et de fermeture des bus. Le mot binaire décodé par l'automate s'appelle une instruction, une suite d'instructions constitue un programme, celui-ci fait fonctionner l'unité centrale.

Il ne faut pas confondre une instruction du programme et une micro-instruction interne à l'unité centrale et ignorée complètement de l'utilisateur. Le contenu du décodeur d'instruction est spécifique au microprocesseur et n'est connu que du fabricant.

1.2.2. *Automatisation de l'unité centrale de base (voir figure 1.5)*

On a donné le nom W à l'accumulateur.

Soient X et Y les données à additionner, et R le résultat de l'addition.

Les instructions nécessaires pour réaliser une addition sont :

```
MOVLW  X ; move (déplacer) la donnée X dans l'accumulateur W
ADDLW  Y ; additionner la donnée Y avec l'accumulateur W.
```

Ces instructions seront écrites en langage mnémonique qui sera ensuite le langage source avec lequel on écrira les programmes.

Ce langage est en fait l'abréviation des mots anglo-saxons.

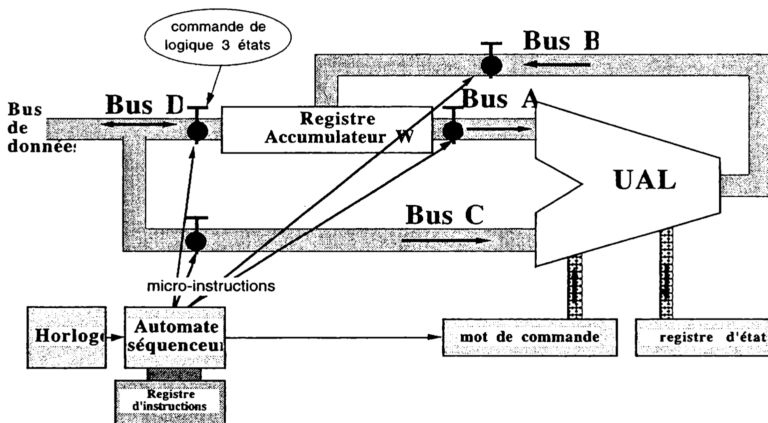


Figure 1.5. *Structure théorique de l'unité centrale*

MOVLW : cette instruction placée dans le registre d'instructions va être décodée par l'automate. L'automate décodeur est séquencé par une horloge généralement pilotée par un quartz. Il va générer des micro-instructions (ouverture et fermeture de bus par exemple) qui vont placer la première donnée X dans l'accumulateur W.

ADDLW : le séquenceur va ensuite chercher l'instruction suivante toujours en suivant le même mode opératoire et la décode. Il prend la donnée Y en mémoire et la place sur la deuxième entrée de l'UAL. Le mot de commande ADDITION est placé sur les entrées de commande de l'UAL, le contenu de l'accumulateur W étant placé sur la première entrée de l'UAL. le résultat R se retrouve immédiatement dans l'accumulateur W.

Cette ébauche de miniprogramme permet de suivre la procédure pour la réalisation d'une addition et de découvrir plus généralement le cheminement des informations sur les bus du microcontrôleur. Toutefois il existe d'autres registres dans un microprocesseur, certains utilisés directement par l'unité centrale, et d'autres par l'utilisateur afin de faciliter la programmation, notamment en permettant l'usage de différents modes d'adressage pour une même instruction (exemple : registre pour l'adressage indirect).

1.2.3. *Spécificité du microcontrôleur PIC*

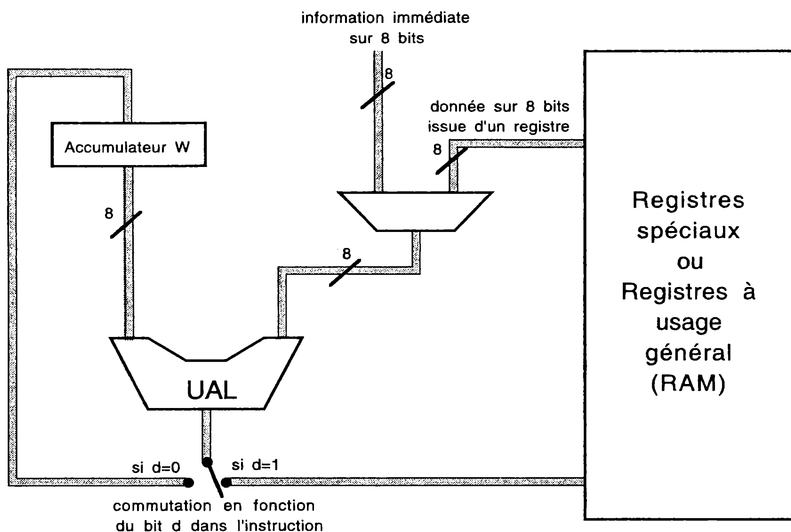


Figure 1.6. *Architecture simplifiée autour de l'UAL d'un PIC*

Les instructions peuvent posséder une ou deux opérandes. Si l'instruction travaille avec une opérande celle-ci se trouve ; soit dans l'accumulateur W, soit dans l'un des registres de la zone RAM. Dans le cas d'une instruction à deux opérandes, l'une est dans l'accumulateur W et l'autre dans l'un des registres.

Le résultat de l'opération à la sortie de l'UAL est dirigé vers l'accumulateur ou vers l'un des registres en fonction d'un bit d contenu dans l'instruction (voir figure 1.6) On se reportera au chapitre 4 concernant les instructions du microcontrôleur.

1.2.4. *Le déroulement d'un programme*

1.2.4.1. *Architecture Harvard et architecture von Neuman*

Dans l'architecture Harvard du microcontrôleur PIC, la mémoire contenant le programme est distincte de la mémoire de données ; l'accès à chacune d'elles se fait par des bus séparés alors que dans l'architecture von Neuman (68HC11), les données et le programme sont chargés depuis la mémoire par le même bus.

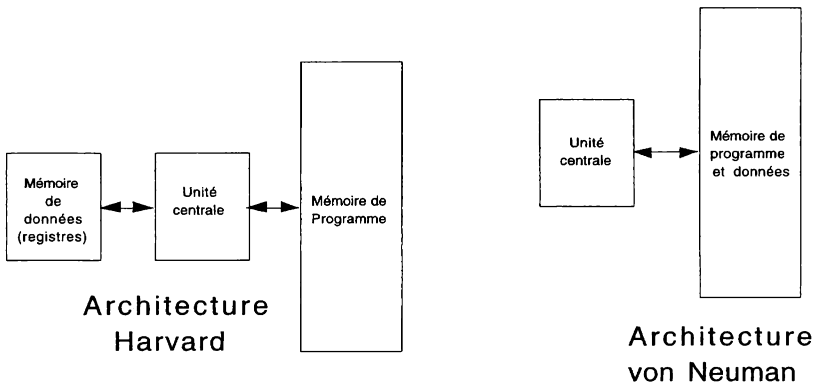


Figure 1.7. *Différence entre architecture von Neuman et architecture Harvard*

Dans un PIC l'instruction écrite sur 14 bits est composée de l'instuction proprement dite et d'une opérande. C'est au niveau du registre d'instruction que la séparation des deux s'effectue (voir figure 1.8.).

L'instruction (opération) est décodée et l'opérande est envoyée par les bus soit à l'entrée de l'UAL dans le cas d'un adressage immédiat (l'opérande est la donnée à traiter), soit elle sert à adresser l'un des registres en RAM (FILE) dans lequel se trouve l'information à traiter (dans le cas d'un adressage direct).

1.2.4.2. *Etude d'une séquence*

```
MOVLW  X ; mettre la donnée X dans l'accumulateur W
ADDLW  Y ; additionner la donnée Y avec le contenu
         ; de l'accumulateur W.
```

A la mise en route de l'unité centrale, une série de micro-instructions va automatiquement charger 0x00 (adresse de la première instruction à éxécuter) dans le registre appelé Compteur de Programme (PC).

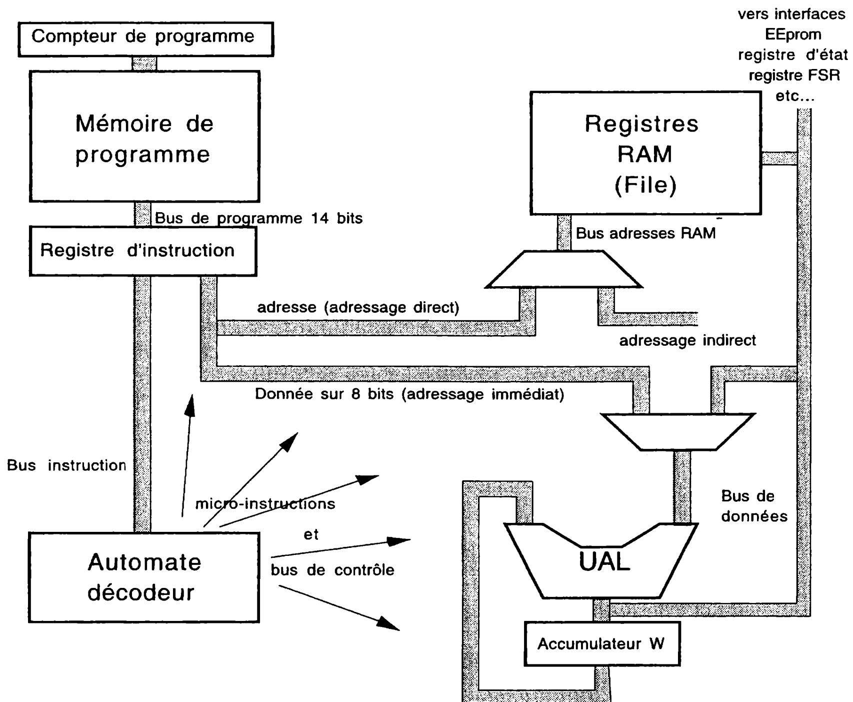
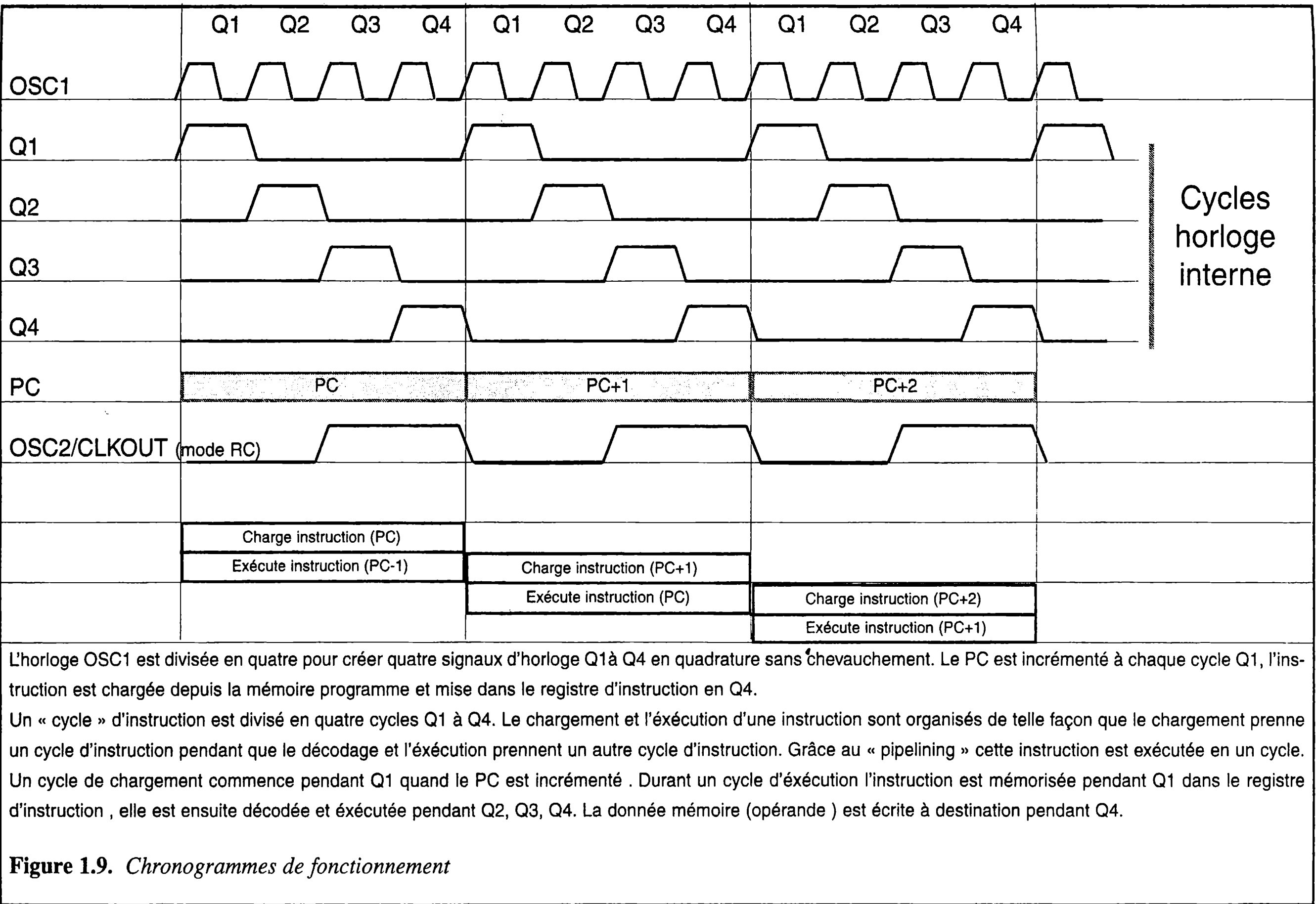


Figure 1.8. *Architecture simplifiée d'un microcontrôleur PIC*

Cette opération que l'on reverra par la suite s'appelle RESET. Ce compteur de programme, comme son nom l'indique, est destiné à contenir l'adresse de l'instruction qui va être exécutée. Il compte les adresses, c'est-à-dire qu'il s'incrémente dès que l'unité centrale a chargé l'instruction complète qu'il pointe en mémoire.

Le compteur de programme est incrémenté pendant l'impulsion d'horloge Q1 (voir figure 1.8).

L'instruction avec son opérande est chargée à partir de la mémoire dans le registre d'instruction par le bus de programme pendant l'impulsion d'horloge Q4 du premier cycle. Ensuite l'automate séquenceur va décoder l'instruction contenue par le registre d'instruction et va générer des micro-instructions qui permettront d'exécuter l'instruction pendant les cycles d'horloge Q1 à Q4 suivants. Le premier travail du



L'horloge OSC1 est divisée en quatre pour créer quatre signaux d'horloge Q1à Q4 en quadrature sans chevauchement. Le PC est incrémenté à chaque cycle Q1, l'instruction est chargée depuis la mémoire programme et mise dans le registre d'instruction en Q4.

Un « cycle » d'instruction est divisé en quatre cycles Q1 à Q4. Le chargement et l'éxécution d'une instruction sont organisés de telle façon que le chargement prenne un cycle d'instruction pendant que le décodage et l'éxécution prennent un autre cycle d'instruction. Grâce au « pipelining » cette instruction est exécutée en un cycle.

Un cycle de chargement commence pendant Q1 quand le PC est incrémenté . Durant un cycle d'éxécution l'instruction est mémorisée pendant Q1 dans le registre d'instruction , elle est ensuite décodée et éxécutée pendant Q2, Q3, Q4. La donnée mémoire (opérande) est écrite à destination pendant Q4.

Figure 1.9. *Chronogrammes de fonctionnement*

décodeur étant de séparer l'instruction de son opérande, de transmettre l'opérande ou donnée vers l'UAL par le bus de données et de transférer l'opération à exécuter à l'UAL.

Pendant le premier cycle Q1 à Q4 il se passe deux choses simultanément :

– l'instruction chargée précédemment est exécutée ;
– une nouvelle instruction est chargée dans le registre d'instruction.

On peut donc dire que le microcontrôleur exécute une instruction par cycle ; de là vient sa plus grande rapidité par rapport à la structure Van Neuman, que l'on rencontre dans d'autres systèmes comme le 68HC11 où une instruction nécessite plusieurs accès mémoire donc plusieurs cycles pour être exécutée.

Dans le cas du PIC dont la structure est appelée HARVARD, un cycle machine est égal à quatre cycles d'horloge du quartz, les cycles Q1 à Q4 sont donc en relation directe avec la fréquence du quartz. Avec un quartz de fréquence 20 MHz le microcontrôleur peut donc réaliser cinq millions d'opérations par seconde.

1.2.4.3. *Cas d'une instruction de branchement*

Si une instruction provoque une modification du contenu du compteur de programme, deux cycles machines sont nécessaires.

EXEMPLE.– L'instruction GOTO

Premier cycle :

– Q1 => décodage de l'instruction,
– Q2 => lecture en immédiat de l'opérande c'est-à-dire du déplacement,
– Q3 => calcul de la nouvelle adresse où on doit se brancher,
– Q4 => écriture de cette nouvelle adresse dans le PC.

Deuxième cycle :

– Q1 => NOP,
– Q2 => NOP,
– Q3 => NOP,
– Q4 => NOP.

Pour l'instruction CALL (appel à un sous-programme), pendant le cycle Q2 du premier cycle d'instruction, il y a lecture du déplacement et en même temps sauvegarde du contenu du compteur de programme dans la PILE.

1.2.5. *Résumé : rôle des composants utilisés*

1.2.5.1. *L'UAL*

Circuit combinatoire permettant les opérations arithmétiques et logiques sur des mots binaires. Ce circuit dispose de deux entrées et d'une sortie, dans la composition d'une unité centrale, sur l'une des entrées de l'UAL on trouve un registre accumulateur W.

1.2.5.2. *L'accumulateur W*

Ce registre sert à stocker temporairement la donnée sur laquelle va travailler l'instruction en cours de déroulement. Ce registre a une capacité en rapport avec la longueur des données que peut traiter l'unité centrale.

1.2.5.3. *Le registre d'instruction*

L'instruction est chargée dans ce registre pendant son exécution. C'est-à-dire pendant son décodage par l'automate décodeur.

1.2.5.4. *Le registre d'état (STATUS)*

Ce registre permet de repérer l'état (niveau logique), à un moment précis, d'une partie bien définie du microprocesseur. Chaque bit est une bascule dont l'état a une signification particulière, il est appelé indicateur ou flag.

1.2.5.5. *L'horloge*

L'horloge permet un cadencement synchrone de l'exécution des séquences de micro-instructions par le décodeur automate.

1.2.5.6. *Le décodeur automate*

Ce composant décode les instructions et génère des séquences de micro-instructions. Il assure le fonctionnement interne de l'unité centrale et notamment le déplacement des données entre les différents éléments.

1.2.5.7. *Le compteur de programme*

Ce registre permet le déroulement chronologique des instructions et désigne (par comptage) les adresses de la mémoire de programme où sont stockées les instructions successives constituant le logiciel.

Processus :

– il pointe l'adresse de la première instruction ;

– il est incrémenté de 1 pendant l'exécution de la première instruction, il pointe alors l'instruction suivante.

Son contenu peut aussi être modifié par l'utilisateur lui-même, mais aussi par le programme grâce à une instruction appelée branchement ou saut.

Chapitre 2
L'unité de calcul

Nous avons vu précédemment le rôle de :

- de l'UAL ;
- du registre d'instruction ;
- du compteur de programme ;
- du registre accumulateur W ;
- de l'horloge.

Le tableau de la page suivante (voir figure 2.1). fait découvrir de nouveaux registres directement accessibles par l'utilisateur, ce sont en réalité des cases mémoires RAM regroupées dans une zone appelée par le constructeur FILE (comme fichier).

Ceux-ci permettent notamment d'utiliser différents procédés de programmation appelés modes d'adressage et aussi de piloter les interfaces

2.1. Définition des registres

Une zone de RAM répartie en deux blocs pour le PIC 16F84 à partir de l'adresse 00h contient des registres de contrôle et d'état des périphériques et des registres du noyau du microprocesseur (STATUS, INDF, SFR, etc). Cette zone de RAM appelée SFR (*Special Fonction Registers*) commence à l'adresse 00h ou 80h selon le bit RP0 du registre STATUS et se termine à l'adresse 0Bh ou 8Bh.

A partir de là, 68 registres GPR (*General Purpose Registers*) sont à la disposition de programmeur pour toutes les opérations dont il a besoin. Cette zone se termine aux adresses 4Fh ou CFh, le reste de la mémoire jusqu'à FFh n'est pas en service.

Les deux blocs sont sélectionnés par le bit RP0 du registre STATUS (si RP0 = 0 on sélectionne la banque 0). Le microcontrôleur 16F877 possède 4 blocs de mémoire, il faut alors programmer un deuxième bit de sélection RP1.

Chacun des registres SFR sera étudié en détail dans la suite du cours en fonction de son utilité.

	Bloc 0	Bloc 1	
$00	Reg. adressage indirect	Reg.adressage indirect	$80
$01	TMRO	option	$81
$02	pcl	pcl	$82
$03	status	status	$83
$04	fsr	fsr	$84
$05	porta	trisa	$85
$06	portb	trisb	$86
$07			$87
$08	eedata	eecon1	$88
$09	eesadr	eecon2	$89
$0A	pclath	pclath	$8A
$0B	intcon	intcon	$8B
$0C $4F	*68 registres en RAM à usage général*	*Cette zone est adressée en même temps que la zone du bloc 0.*	$8C $CF
$50 - - $7F	Zone non utilisable : donne 0 en lecture		$D0 - - $FF

Figure 2.1. *Les registres du PIC 16F84*

2.2. Les registres « PCL » et « PCLATH »

Le compteur de programme ou compteur ordinal nommé PC permet de pointer sur la prochaine instruction à exécuter, la structure du microcontrôleur PIC impose un fonctionnement particulier à ce compteur de programme.

Le PC complet étant d'office sur 13 bits (mémoire maximum d'un PIC : 8 Ko) et les registres du PIC ne faisant que 8 bits, il faut donc 2 registres pour reconstituer une adresse dans le PC.

Le premier accessible en lecture et écriture contient l'adresse basse du PC sur 8 bits, il est appelé PCL (PC *Low*), le deuxième s'appelle PCLATH (PC LATch *counter High*).

Lors d'un saut, le contenu du PC est chargé directement avec les 11 bits de l'adresse de destination contenus dans l'instruction en elle-même. Les 2 bits manquants sont extraits du registre PCLATH. Ces bits b3 et b4 qui doivent être positionnés par l'utilisateur, sont placés directement dans les bits b11 et b12 du PC afin de compléter l'adresse de destination. Comme le PIC 16F84 ne gère que 1K de mémoire programme, nous n'aurons pas besoin de ce registre.

En cas de modification du PCL directement par l'utilisateur, le PCL (8 bits) est chargé dans le compteur de programme et est complété par les 5 bits du registre PCLATH. Les bits b2, b3 et b4 de PCLATH sont inutilisés pour le PIC16F84.

Le contenu du PC peut être modifié par programme (exemple : ADDWL PCL).

2.3. Le registre « W »

Ce registre de travail que l'on peut appeler accumulateur est un registre utilisé pour réaliser des calculs. Le résultat d'un calcul peut être sauvegardé dans un emplacement RAM (f) ou dans le registre de travail (w). Le choix se fait en fonction de la valeur du bit d de l'instruction.

2.4. Le registre « STATUS » ou registre d'état

STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
03h ou 83h	R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x

R/W veut dire que ce bit peut être lu et ecrit. R veut dire qu'il ne peut être que lu.
Au moment du reset il est mis à 1 ou à 0.

Il renseigne sur les résultats des calculs et opérations effectués. Il est donc utilisé pour les tests et les branchements conditionnels.

C : *Carry* (retenue ou report)

Il n'est modifié que par les instructions ADDWF, ADDLW, SUBLW et SUBWF.

Il passe à 1 quand une retenue apparaît sur le calcul du bit le plus significatif. Ce bit est aussi utilisé pour les instructions de rotation. Pour une soustraction la polarité est inversée, la soustraction est exécutée comme l'addition du complément à deux du second opérande.

DC : *Digit Carry*

Ce bit est utilisé principalement lorsque l'on travaille avec des nombres BCD : il indique un report ou retenue du bit 3 vers le bit 4.

Z : *Zero* – indicateur de zéro

Ce bit est positionné à 1 si le résultat de la dernière opération logique ou arithmétique vaut 0. Z n'est positionné que pour certaines instructions (voir tableau page 56).

Après une addition (ADDWF), Z sera modifié si le résultat est nul mais après un chargement MOVWF avec une donnée nulle, Z n'est pas modifié.

PD : *Power-down*

Ce bit est actif au niveau zéro. Il est mis à 1 à la mise sous tension (POR) mais n'est pas modifié par le *master-clear* (MCLR).

PD est mis à 0 par l'instruction SLEEP (état de basse consommation).

PD est mis à 1 par l'instruction CLRWDT (mise à 0 du chien de garde).

TO : *Time-out bit*

Ce bit est actif au niveau zéro. Ce bit est mis à 1 à la mise sous tension (POR), l'instruction CLRWDT (mise à 0 du chien de garde) ou l'instruction SLEEP. Ce bit est mis à 0 par le débordement du chien de garde, il peut indiquer si le redémarrage suit un arrêt provoqué par débordement du compteur du chien de garde). Il n'est pas modifié par le MCLR.

RP0, RP1 : *Register Bank Select*

Ces deux bits permettent d'indiquer dans quel bloc de RAM on veut travailler. Si RP0 et RP1 sont à 0 on utilise le bloc 0.

RP1 est inutilisé pour le PIC 16F84, il doit être laissé à 0 pour garantir la portabilité du programme dans un microcontrôleur plus performant (avec plus de 2 blocs mémoire comme le Pic 16F877).

IRP : *Indirect RP*

Permet de choisir quel bloc de RAM on veut utiliser en cas d'adressage indirect.

IRP n'est pas utilisé dans le PIC 16F84, il doit donc être maintenu à 0.

2.5. Utilisation du registre d'état

Les indicateurs du registre d'état sont positionnés automatiquement par l'unité centrale lors de certaines opérations mais peuvent aussi être manipulés par l'utilisateur grâce à des instructions spécifiques. Ces indicateurs peuvent être lus par l'utilisateur mais sont utilisés automatiquement par l'unité centrale pour des opérations de branchement conditionnel.

EXEMPLE.– DECFSZ décrémente un registre et effectue un test sur le bit Z du registre d'état.

Si la décrémentation donne un résultat nul le bit Z passe à 1.

Un branchement ou saut au-dessus de l'instruction qui suit ne s'effectue que si le bit Z est à 1. Si Z est à 0 c'est l'instruction qui suit qui est exécutée, généralement il s'agit d'une instruction de branchement.

Le « test » du bit Z est complètement transparent pour l'utilisateur.

EXEMPLE.–

```
decfz TOTO      ; décrémente le registre TOTO et teste
                ; le bit Z
goto bou        ; si Z = 0 l'instruction goto est exécutée
movlw 0x07      ; si Z= 1 l'instruction goto est ignorée,
                ; movlw est exécutée
```

2.6. Fonctionnement de la pile

C'est une zone de mémoire RAM spécifique de format 13 bits située en dehors de la zone programme, utilisée pour stocker temporairement des adresses. Le pointeur de pile est décrémenté automatiquement à chaque adresse placée dans la pile, et est incrémenté à chaque adresse déchargée, il ne peut être ni lu ni écrit. Cette pile ne dispose que de 8 emplacements.

Ce mécanisme est de type LIFO (*Last Input, First Output* => dernier entré, premier sorti).

Contrairement à d'autres microprocesseurs, les instructions PUSH et PULL qui permettaient au programmeur de placer dans la pile des données ou en retirer, n'existent pas.

2.6.1. *Usage de la pile lors de l'appel d'un sous-programme*

Alors que l'unité centrale est en cours d'exécution d'un programme appelé principal, elle rencontre l'instruction de branchement à un autre programme : CALL. Celui-ci est appelé par opposition sous-programme ou programme secondaire. Une fois ce sous-programme exécuté, celui-ci se terminant obligatoirement par une instruction de retour (RETURN), l'unité centrale reprendra le cours du programme principal quitté précédemment.

Afin qu'elle puisse le reprendre à l'endroit où elle l'avait quitté, il faut lui mettre en mémoire l'adresse de retour. Nous savons que le compteur de programme est incrémenté dès qu'une instruction a été chargée dans le registre d'instruction. Dans

ce cas dès que l'instruction de branchement est entièrement lue, le compteur de programme contient l'adresse de l'instruction suivante. Celle-ci est en fait l'endroit où doit reprendre le programme principal au retour du sous-programme.

EXEMPLE.–

```
h0050    ******
h0051    CALL  toto
h0052    ******
```

A l'adresse h0051 (valeur héxa-décimale) on trouve l'instruction de branchement CALL, l'endroit où doit s'effectuer le branchement est symbolisé par l'étiquette toto. Quand le sous-programme sera terminé il faudra reprendre le programme principal en h0052, c'est cette adresse de retour, contenue à cet instant dans le PC, qui sera sauvegardée dans la pile. Le pointeur de pile sera décrémenté de un. Cette mise en mémoire dans la pile du contenu du PC (adresse de retour) est exécutée automatiquement par l'unité centrale.

A la fin du sous-programme l'instruction RETURN provoque l'opération inverse, le compteur de programme sera rechargé avec l'adresse h0052 et le pointeur de pile sera incrémenté de un. Le programme principal reprend là où il a été quitté.

2.6.2. *Usage de la pile en cas d'interruption*

On va retrouver le même mécanisme (LIFO) puisque l'interruption déclenche le déroulement d'un sous-programme (voir chapitre 6).

Dans un sous-programme d'interruption demandé par un périphérique ou un signal extérieur de façon asynchrone, les registres de l'unité centrale (W et STATUS) sont appelés à être modifiés (calculs, etc.). Il est donc souvent nécessaire de les sauvegarder afin de pouvoir, après ce sous-programme, reprendre les opérations qui étaient en cours dans les mêmes conditions.

Le programmeur devra donc sauvegarder ces registres dans des cases mémoires RAM (registres) qui ne seront pas modifiées par le sous-programme d'interruption.

Un sous-programme d'interruption se termine par l'instruction RETFIE qui provoque la reprise du programme principal à l'endroit où il a été quitté, le programmeur devra prévoir la récupération des registres qu'il a sauvegardés.

La gestion du pointeur de PILE est automatique et complètement transparente pour l'utilisateur, toutefois, le nombre d'emplacements mémoire est limité à 8 pour le PIC 16F84, il faut donc faire attention au nombre maximum d'imbrications de sous-programmes sous peine de plantage car après 8 imbrications le pointeur de PILE reprend sa valeur initiale et les valeurs de retour des sous-progammes sont perdues.

2.6.3. *La configuration du microcontrôleur*

CONFIG	CP	CP	CP	CP	CP	CP	CP	CP	CP	CP	PWRTE	WDTE	FOSC1	FOSC0
	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u

R : lisible - P : programmable - u : inchangé au POR

Un registre à l'adresse 2007h situé après la zone programme permet la configuration du microcontrôleur. Toutefois il n'est accessible que pendant la phase de programmation du programme dans le microcontrôleur et ne peut être utilisé durant le programme.

C'est un registre sur 13 bits dont une partie contient le code de protection du microcontrôleur.

2.6.3.1. *Contenu du registre CONFIG*

FOSC0-FOSC1 : choix de l'oscillateur

Fosc1	Fosc0	Type d'oscillateur
0	0	Oscillateur avec réseau résistance capacité. Fmax : 4 MHz.
0	1	Oscillateur XT à quartz ou oscillateur extérieur. Fmax : 4 MHz
1	0	Oscillateur HS à quartz grande vitesse. Fmax : 10MHz.
1	1	Oscillateur LP à quartz basse consommation. Fmax : 200KHz

L'horloge du microcontrôleur peut utiliser différents systèmes, il faut configurer le PIC en fonction du câblage réalisé et de ses caractéristiques de fonctionnement.

Le microcontrôleur ne comprend pas de mémoire dynamique, son horloge peut donc théoriquement descendre à 0.

WDTE (*Watch Dog Timer Enable*) : met en service le chien de garde (Watchdog) quand il est à 1.

PWRTE (*PoWeR Timer Enable*) : met en service le timer de reset de mise sous tension quand il est à 1.

2.6.3.2. *Configuration*

Pour configurer le microcontrôleur il faut utiliser la directive CONFIG. Celle-ci fait appel à des étiquettes contenues dans le fichier « include » : P16F84.INC.

Les informations de configuration seront intégrées dans le fichier de codes machine « .hex » qui sera programmé dans le PIC par le programmateur.

EXEMPLE.–

__CONFIG _CP_OFF & _WDT_ON & PWRTE_ON & _HS_OSC

CONFIG est la pseudo-instruction ou directive.

Chaque bit est suivi de ON ou OFF:

_CP_OFF met le bit CP à 1 ; il n'y a pas protection

_PWRTE_ON met le bit PWRTE à 0 ; le timer est mis en service.

Il est possible aussi de programmer ce registre en écrivant dans le source du programme :

```
ORG 0x2007
DA B'11111111111110'
```

Dans cet exemple : il n'y a pas de code de protection, le *watchdog* est en service, le *timer* du *reset* est en service et c'est un oscillateur à quartz grande vitesse qui est utilisé pour l'horloge.

Chapitre 3
La programmation

3.1. Le langage source assembleur

Nous avons vu précédemment comment écrire le chargement d'un accumulateur, MOVLW X veut dire charger l'accumulateur W avec la donnée X.

MOVLW est l'abréviation de MOVe Literal to W.

Le langage source assembleur est en quelque sorte l'écriture abrégée des commandes en langue anglaise, il est dédié à la famille du microprocesseur utilisé. Dans notre cas il s'agit bien entendu de la famille PIC de MICROCHIP. Toutefois, le chargement de l'accumulateur peut se faire avec une donnée dont l'origine peut être diverse : donnée déjà en mémoire dans un registre, ou donnée choisie directement par le programmeur. Il est donc indispensable de bien distinguer ces différentes sources, cela peut se faire grâce à différents modes que l'on appelle les modes d'adressage.

3.1.1. *Conventions d'écriture*

Le langage permet à l'utilisateur l'écriture des nombres sous différentes formes, il faut toutefois les préciser :

– un nombre décimal n'est précédé d'aucun symbole : on écrit 5, ou précédé d'un D : D'5';

– un nombre hexadécimal est précédé de la lettre H : on écrit H'9D' ou 0x09D ;

– un nombre octal est précédé de O : on écrit O'72';

– un nombre binaire est précédé de B : on écrit B'10010011';

– un caractère ASCII : on écrit A'x', x étant le caractère.

3.1.2. *Définition*

On appelle adresse effective, l'adresse qui est directement manipulée par l'instruction.

EXEMPLE.– MOVWF 0x050 veut dire mettre en mémoire l'accumulateur W dans la case mémoire dont l'adresse est 50h, cette adresse est directement utilisée par l'instruction elle est donc l'adresse effective. Ceci est défini par opposition à certaine structure d'adressage où l'adresse effective est déterminée par l'instruction à partir d'une autre adresse.

3.1.3. *Structure d'une instruction en langage assembleur*

Une instruction doit être rédigée (cadrée) de façon très précise : elle est constituée de trois champs d'édition. Un quatrième champ facultatif, mais pourtant vivement conseillé, contient les commentaires du programmeur.

– *le champ d'adresse d'instruction* contient l'adresse où se trouve l'instruction dans le programme c'est-à-dire en mémoire, cette adresse n'est pas à écrire dans le source car on utilise des étiquettes. Une étiquette ou label doit commencer par une lettre ou par un tiret bas « _ » ;

– *le champ code d'instruction* comporte un symbole qui représente sous forme mnémonique la fonction de l'instruction (exemple : MOVLW) ou une directive d'assemblage ou encore une macro ;

– *le champ d'adresse d'opérande* contient, quand l'instruction comprend une référence mémoire, le nom associé à cette position mémoire (étiquette) ou l'adresse proprement dite de l'opérande. Si il y a plusieurs opérandes elles sont séparées par une virgule.

REMARQUE.– Quand l'adressage est du type immédiat ce champ contient la valeur de l'opérande.

– *le champ commentaire* peut commencer à la première colonne par un point virgule ou après le champ d'opérande mais séparé par un point virgule.

EXEMPLE.–

```
Champ        Champ code      Champ d'adresse    Commentaires
d'adresse    d'instruction   d'opérande
000          ADDLW           4                  ;additionne 4 au
                                                ;contenu de W
```

Dans cet exemple, on additionne la valeur de l'opérande (4) à l'accumulateur, l'instruction est placée à l'adresse 000.

Le logiciel assembleur reconnait automatiquement le premier mot de la ligne de commande comme un label ou une instruction.

3.2. Les différents modes d'adressage

3.2.1. *Adressage immédiat*

Le code opératoire est suivi par l'opérande et ne fait appel à aucune adresse. L'opérande est la valeur à utiliser par l'instruction, le constructeur l'appelle "literal" et le note K.

EXEMPLE.– MOVLW 0x34 ; charge la valeur hexadécimale 34 dans l'accumulateur W avec MOV pour move, L pour literal (immédiat) et W pour l'accumulateur w.

3.2.2. *Adressage direct*

Le code opératoire est suivi d'un octet non signé qui est l'adresse effective de l'opérande à traiter. L'opérande est l'adresse de la valeur à utiliser par l'instruction.

EXEMPLE.– MOVF 0x5,W ; charge l'accumulateur W avec la donnée située
; dans la case mémoire 5.
avec MOV pour move et F pour File (registre).

L'adresse écrite sur 7 bits est complétée, en adressage DIRECT, par les bits RP0 et RP1 du registre STATUS (RP1 est inutile dans le PIC 16F84 car il y a seulement 2 blocs mémoire ; on le laisse à 0).

EXEMPLE.– Si RP0 est à 0, l'adresse 5 est celle du PORTA dans le bloc mémoire 0 et si RP0 est à 1, l'adresse 5 devient 85h, c'est l'adresse du registre TRISA dans le bloc 1.

3.2.3. *Adressage indirect*

L'adresse de l'opérande est cette fois écrite avec un pointeur d'adresse. L'adressage indirect utilise deux registres du microcontrôleur : INDF et FSR.

INDF (INDirect File) est à l'adresse 0h. Ce registre n'existe que pour accéder au à la donnée pointée par le registre pointeur d'adresse FSR. Ce registre FSR se trouve à l'adresse 0x04 ou 0x084.

Pour accéder à la donnée contenue dans la case mémoire X, il suffit d'écrire dans le registre FSR l'adresse X. En lisant le registre INDF, on trouve la valeur contenue dans la case mémoire X.

Si FSR contient 0, la lecture de INDF donnera 0.

REMARQUE.– Le contenu du registre FSR pointe sur une adresse écrite sur 8 bits. Or, sur certains PICs, la zone RAM contient 4 blocs (16F877). L'adresse complète est donc une adresse sur 9 bits réalisée par l'ajout du bit IRP du registre STATUS (inutile dans le 16F84 ; on le laisse à 0).

EXEMPLE 1.– Le registre à l'adresse 7 contient 0x0A.

Pour lire le contenu de cette case il suffit d'écrire 7 dans le FSR, puis de lire le contenu de INDF.

En incrémentant le contenu de FSR, on accèdera au contenu de la case mémoire 8.

EXEMPLE 2.– Accèder au contenu du registre x ; l'adresse du registre x est 0x0A

```
    movlw 0x50      ; chargeons 50h dans W
    movwf x         ; plaçons 50h dans x
    movlw 0x0A      ; on charge l'ADRESSE de x dans W
    movwf FSR       ; on place l'adresse de destination dans FSR;
                    ; FSR POINTE sur x
    movf INDF,w     ; charger le CONTENU de INDF dans W.
```

Après ce programme W contient le contenu du registre x soit 0x50.

EXEMPLE 3.– Remettre à 0 des registres entre l'adresse h15 et l'adresse h1F ;

```
    MOVLW 0x15      ; charge h15 dans W, adressage immédiat
    MOVWF FSR       ; transfert de W dans FSR pour former
                    ; l'adresse (adressage  direct)
Suite CLRF INDF     ; mise à 0 de INDF donc de la case mémoire
                    ; pointée (adressage direct)
    INCF FSR        ; incrémentation du pointeur adressage direct
    BTFSS FSR,5     ; test du bit  5 (0010 0000) et saut à FIN si
                    ; le bit est à 1
    GOTO suite      ; tant que le bit est à 0 on boucle à suite
                    ; pour recommencer.
FIN ======
```

3.2.4. *Adressage relatif*

Ce mode d'adressage s'applique aux opérations de branchement ou de saut CALL et GOTO. Il permet de faire exécuter des instructions placées dans une autre portion de l'espace mémoire de programme que celle où se déroule le programme en cours. Pour cela il modifie le contenu du compteur de programme sachant que lors de l'exécution d'une instruction le PC pointe sur l'instruction suivante.

Avec l'adressage relatif, l'adresse de la prochaine instruction à exécuter sera calculée en additionnant le contenu du compteur de programme avec la valeur du déplacement additionnée de un.

Le déplacement est un octet en binaire signé (positif ou négatif) généralement calculé en fonction d'une étiquette.

EXEMPLE.–

```
BOU  decfsz T1,f         ; decremente T1, si resultat = 0
                         ; saute l'instruction suivante
     goto BOU            ; branche a BOU
```

Pendant l'assemblage l'étiquette BOU qui suit goto sera remplacée par la valeur du déplacement qui additionnée au contenu du compteur de programme permettra à l'unité centrale d'effectuer l'instruction decfsz T1,f.

3.2.5. *Mode manipulation de bit*

Il permet de modifier un bit dans un registre. L'instruction a besoin de l'adresse du registre et de la position du bit.

BCF veut dire Bit Clear dans File (registre) et BSF veut dire Bit Set dans File.

EXEMPLE.–

```
BCF PORTA,3   ; met à 0 le bit 3 du registre PORTA.
BSF PORTB,5   ; met à 1 le bit 5 du PORTB
```

3.2.6. *Mode test de bit et branchement*

Ce mode permet de tester la valeur d'un bit d'une donnée en mémoire et d'effectuer ou non un branchement en fonction de l'état de ce bit.

EXEMPLE.–

```
BOU btfss INTCON,T0IF   ; test si TMR0 a debordé ?
 goto BOU             ; si non recommence
 bcf VERIN            ; si oui avance verin
```

Dans cet exemple l'instruction BTFSS teste la valeur du bit TOIF du registre INTCON et effectue ou non un branchement à la deuxième ligne qui suit (bcf VERIN).

3.3. La réalisation d'un programme

Nous avons vu précédemment qu'un programme est constitué d'une suite d'instructions en code machine. Chacune des instructions a pour but de prévoir et de dicter pas à pas tous les détails de fonctionnement et de contrôle que la machine doit exécuter.

3.3.1. *Comment écrire un programme ?*

3.3.1.1. *Le langage d'assemblage ou source assembleur*

Ce langage utilise des symboles mnémoniques pour représenter les codes opératoires. Par exemple : MOVF, XORLW, BTFSS, etc.

Le texte source (logiciel rédigé en langage d'assemblage) est écrit à l'aide d'un traitement de texte ou d'un éditeur de texte sur ordinateur. Ce texte est ensuite traité par un autre logiciel appelé assembleur qui traduit le texte en codes machine.

Pour que cet assembleur puisse travailler, il est nécessaire que le texte édité ait une structure très précise, il faut aussi y inclure, en plus des instructions du programme à l'étude, des informations appelées pseudo-instructions ou encore directives d'assemblage. Celles-ci sont utiles uniquement à l'assembleur et ne font pas partie du programme.

Exemple de source assembleur :

```
 titre 'toto'
 list p=16F84
 include "p16F84.inc"

#define VERIN PORTA,0
#define FINCOURS PORTA,1

OPTIONVAL equ B'00110000'    ; reglages
INTMASK equ B'00000000'      ; interruptions
NBPIECE equ 2

 org 0x00
 goto START

START bsf STATUS,RP0     ; passe en banque 1
 movlw OPTIONVAL
 movwf OPTION_REG        ; mise en place des reglages
 movlw INTMASK
 movwf INTCON            ; masque interruptions
 movlw B'00010010'
```

```
 movwf TRISA            ; initialise port A
BOU2
 bcf STATUS,RP0         ; passe en banque 0
 movlw 256-NBPIECE      ; mise en place
 movwf TMR0             ; du nombre de pieces
 bcf INTCON,T0IF        ; RAZ du flag
 bsf VERIN              ; verin en position initiale
BOU btfss INTCON,T0IF   ; test TMR0 a deborde ?
 goto BOU               ; si non recommence
 bcf VERIN              ; si oui avance verin
BOU3 btfss FINCOURS     ; verin en fin de course ?
 goto BOU3              ; si non attendre
 goto BOU2              ; si oui nouveau cycle
 END
```

3.3.2. *L'assemblage*

La transformation du texte en programme, que l'on appelle assemblage, s'effectue en deux temps appelés passes.

Au cours de la première passe le logiciel appelé assembleur, reconnaît chaque symbole d'instruction (mnémonique), en déduit le code machine correspondant.

Après cette première passe, l'assembleur a défini une table des symbôles :

`tempo 0001`

`BOU 0006` (voir exemple en fin de chapitre)

Pendant la seconde passe, cette table des symbôles est utilisée pour achever la traduction. Il faut comprendre qu'un symbôle peut être défini à la fin du texte source et utilisé en début de programme. L'assembleur génère alors le code machine comme une suite d'octets qui pourront ensuite être mis en mémoire, constituant ainsi le programme éxecutable (fichier .hex).

Au cours de son travail l'assembleur va signaler à l'utilisateur toutes les erreurs de syntaxe ou de logique qui l'empêchent de terminer l'assemblage. Il signalera aussi par des avertissements (Warnings) les ambiguités qui risquent de poser problème.

Tout ceci permet à l'opérateur de reprendre son texte source et de le corriger avant un nouvel assemblage.

Une fois l'assemblage terminé l'utilisateur dispose de plusieurs fichiers dans l'ordinateur :

– le fichier source qu'il a édité et qu'il pourra modifier par la suite ;

– le fichier *listing* qui contient toutes les informations nécessaires à la relecture et à la compréhension du programme ;

– le fichier « .hex » qui contient les codes machines exécutés par le microprocesseur est stocké sous forme « texte » sur le disque de l'ordinateur, il contient toutes les informations nécéssaires à son implantation dans la mémoire programme du PIC (ce fichier est étudié en détail à la fin du chapitre) ;

– d'autres fichiers de travail générés par l'assembleur notamment un fichier détaillant les erreurs et avertissements qui subsistent dans la source.

3.3.3. *Règles de rédaction en langage d'assemblage*

3.3.3.1. *Les pseudo-instructions ou directives d'assemblage*

Les directives ne sont utiles qu'au logiciel d'assemblage, elles ne font pas partie du programme.

La directive ORG précise à l'assembleur où sont implantés les codes machines dans la mémoire de programme. Elle s'écrit après une tabulation (un espace minimum). On peut placer plusieurs ORG dans un programme.

Quand il s'agit d'un début de programme, l'assembleur met la première instruction qui suit à l'adresse origine (fixée par ORG).

Après un RESET, un PIC démarre toujours à l'adresse 00, on utilise l'instruction goto pour faire démarrer le programme à un autre emplacement (START est l'étiquette indiquant le début du programme).

EXEMPLE.–

```
ORG 0x00
goto START
```

La directive LIST détermine le type de microcontrôleur utilisé.

EXEMPLE.–

```
List P=16F84
```

La directive END précise où doit s'arrêter l'assemblage, les instructions situées après sont ignorées. En aucun cas cette directive ne signifie la fin du programme.

La directive CONFIG permet de définir les paramètres de fonctionnement du PIC. Pour cela il suffit de positionner la valeur de chacun des cinq bits du registre de configuration. Ce registre de configuration se situe à l'adresse 2007h et n'est pas

accessible directement par programme. Pour le programmer on utilise la directive CONFIG qui n'intervient qu'avec le programmateur.

Exemple de notation :

__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC

Chaque définition est séparée par le symbôle &.

Dans l'exemple ci-dessus le bit CP est à 0, le bit WDT (watchdog) est à 0, le bit PWRTE est à 1 et le terme HS définit le type d'horloge utilisé.

Cette directive n'est utile que pour modifier la configuration par défaut du microcontrôleur.

3.3.3.2. *Les assignations*

Une assignation se comporte comme une simple substitution. Au moment de l'assemblage, chaque fois que l'assembleur va trouver une étiquette avec assignation, il la remplacera automatiquement par sa valeur.

Ceci permet de faciliter la *maintenance* du programme. Il est plus simple de retenir dans un programme la valeur « table » que de manipuler la valeur hexadécimale correspondante 0x30 et quand on change la valeur d'une assignation, le changement s'effectue pour toutes les variables correspondantes utilisées dans le programme.

Exemple d'assignation :

```
Table      EQU      0x30
```

3.3.3.3. *Les fichiers « include »*

La directive « include » permet d'introduire dans le programme les *assignations* contenues dans le fichier P16F84.INC, c'est-à-dire toutes les *constantes* utiles du microcontrôleur.

Par exemple :

```
include "p16F84.inc"
```

Cette directive permet d'introduire toutes les adresses utiles d'un seul coup sans avoir à les mémoriser et à les écrire.

Extrait d'un fichier p16F84.inc :

```
PCL       EQU  H'0002'
STATUS    EQU  H'0003'
FSR       EQU  H'0004'
PORTA     EQU  H'0005'
PORTB     EQU  H'0006'
EEDATA    EQU  H'0008'
EEADR     EQU  H'0009'
PCLATH    EQU  H'000A'
INTCON    EQU  H'000B'
```

3.3.3.4. *Les définitions*

Une définition « #define » fonctionne comme une *assignation.* Les assignations sont utilisées pour les valeurs, et les définitions pour remplacer un texte plus complexe ou le résultat d'un calcul.

La directive #DEFINE, écrite en bordure de marge est suivie par l'étiquette, puis la chaîne à substituer.

Par exemple :

```
#DEFINE  led        PORTB,2
```

Ceci veut dire que l'étiquette « led » désigne le bit 2 du port B.

3.4. Les macros

Une macro permet de nommer une séquence d'instructions souvent utilisée et qui peut être inserée dans un programme par simple appel de son nom. Elle simplifie l'écriture et apparaît comme une instruction.

Elle se compose d'un nom (étiquette) écrit en première colonne, puis de la directive « macro ». A la ligne suivante on place le logiciel qui constitue la macro. La macro se termine par la directice « endm » (*end of macro*).

Exemple de définition :

```
Alum     macro
         bsf led (mise à 1 du bit correspondant à led)
         endm
```

Exemple d'utilisation :

A chaque fois que la macro Alum est rencontrée, elle est remplacée par la ligne :

```
bsf led
```

LED étant la ligne 2 du port B (voir la directive #define), la macro met à un cette ligne donc allume la led.

Macro avec passage de variable

Une macro peut aussi servir à créer des instructions qui n'existent pas comme par exemple l'instruction MOVLF f,k qui charge une donnée k dans un registre f.

Création de la macro :

```
movlf macro f,k         ; movlf est le nom de la macro, f et k
                        ; sont les paramètres.
    movlw k             ; charge k dans w
    movwf f             ; stocke w dans f
    endm                ; fin de macro
```

L'appel de la macro se fait alors de la façon suivante : on écrit l'appel de la macro comme si il s'agissait d'une instruction en définissant alors les deux paramètres.

```
    movlf PORTB,0x55    ; charge 55h dans le PORTB
```

3.5. La zone des variables

Cette directive CBLOCK permet de définir l'adresse du début d'une zone de variables.

EXEMPLE.–

```
    CBLOCK 0x0C     ; début de la zone de variables
```

Ensuite, il suffit d'écrire le nom de la variable suivi du signe « : » et de la taille utilisée (nombre de registres à usage général).

```
    cmpt : 1        ; zone de 1 octet
    table : 8       ; zone de 8 octets
```

La fin de la zone en cours est définie par la directive :

```
    ENDC            ; Fin de la zone
```

3.6. Les étiquettes

Afin de faciliter la relecture et aussi de permettre la modification multiple, plus facile, de données ou d'adresses, l'utilisateur a intérêt à utiliser des étiquettes

(labels). Une étiquette permet aussi de retrouver un emplacement mémoire utilisé par exemple pour un branchement ou un saut. L'assembleur remplace l'étiquette par sa valeur ou par l'adresse du programme où elle se trouve.

Règles :
- le premier caractère doit être une lettre ou un tiret bas « _ » ;
- elle est alphanumérique c'est-à-dire qu'elle peut comporter des chiffres ;
- elle ne peut être définie qu'une seule fois ;
- elle est écrite en première colonne du source ;
- elle ne peut faire plus de 32 caractères.

Exemple :
```
DEBUT clrf PORTA
      "
      "
      "
      goto DEBUT
```

« Debut » est une étiquette qui permet à l'assembleur de calculer le déplacement nécessaire au branchement inconditionnel « goto ».

3.7. Les commentaires

L'utilisateur, pour faciliter la relecture, à long terme, de son texte source ou du listing d'assemblage, a intérêt à y introduire des commentaires :

- soit entre les lignes du programme, dans ce cas il commence chaque ligne par un point virgule {;} en bordure de marge. Exemple :
```
; ceci est un commentaire
```
- soit après une instruction.

Exemple :
```
decfsz comptl,f    ; décrémenter le compteur comptl
```

3.8. Comment bâtir un programme ?

Avant de démarrer l'écriture d'un programme il faut :

- avoir bien compris le problème à résoudre et savoir écrire la solution sous forme d'organigramme compatible avec le microprocesseur utilisé ou encore savoir l'écrire sous forme algorithmique ;

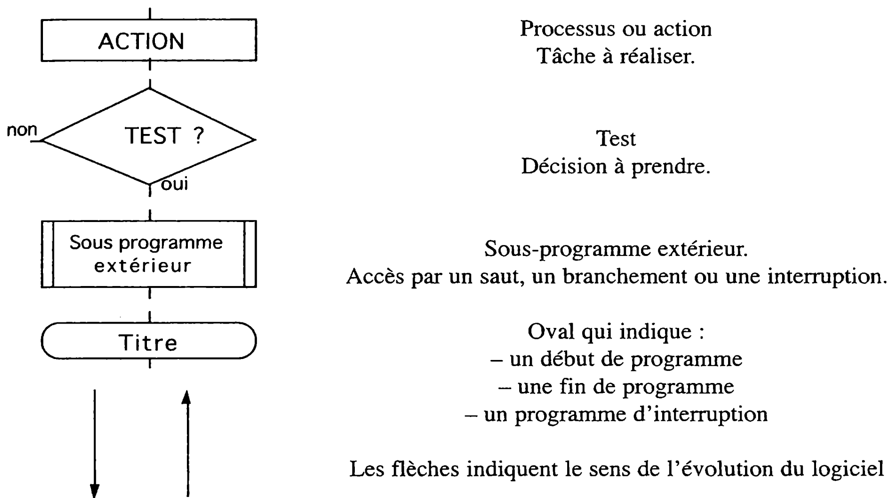
- savoir choisir la méthode la mieux adaptée pour la machine choisie ;

– savoir mettre en application la méthode par un choix judicieux et bien structuré de séquences d'instructions (le programme par lui-même) ;

– savoir vérifier l'exactitude du programme.

REMARQUE.– Tous les algorithmes sont codables mais l'inverse n'est pas vrai.

3.8.1. *Définition de l'organigramme* (flow chart)



3.8.1.1. *Symboles utilisés*

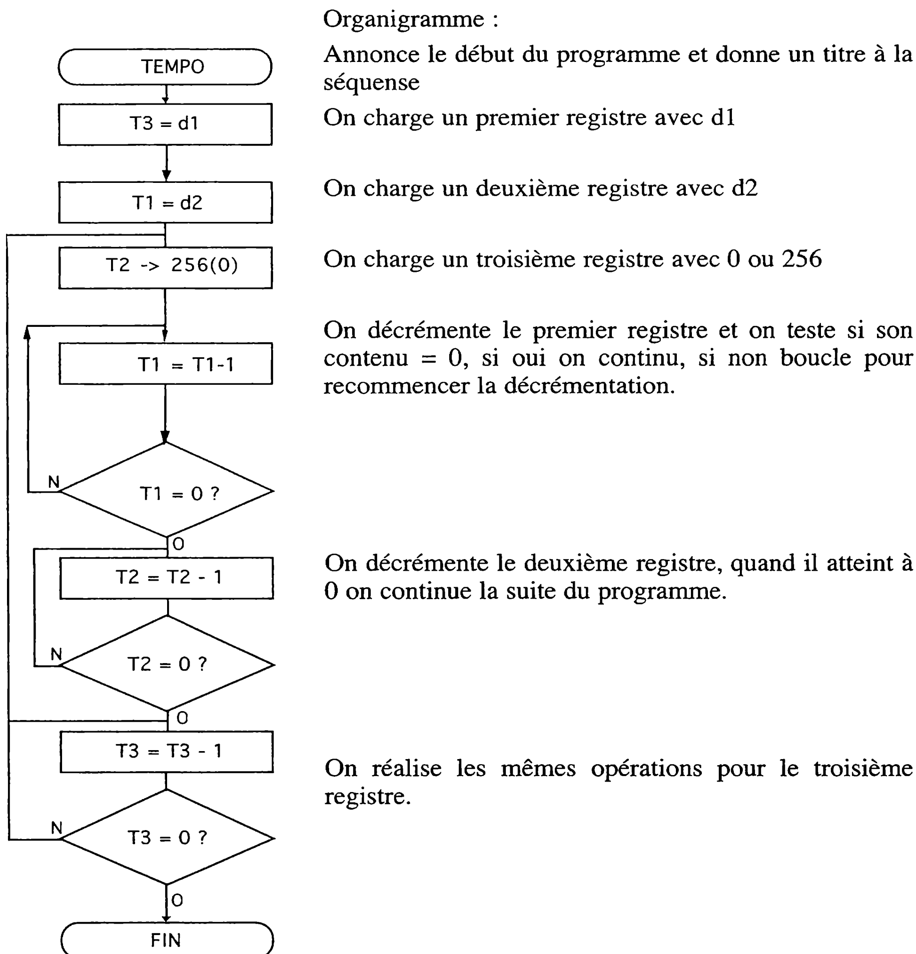
L'organigramme est une représentation graphique illustrant les étapes et les décisions logiques qui sont nécessaires pour accomplir une tâche spécifique (voir figure ci-dessus).

3.8.1.2. *Exemple*

Réalisation d'un intervalle de temps par programme.

Sachant que chaque instruction nécessite un temps d'exécution que l'on peut chiffrer en cycles machine, on peut utiliser ces délais (perte de temps) pour réaliser une temporisation. Toutefois la vitesse d'éxécution 1 cycle par instruction et 2 cycles pour les branchements et la dimension des registres (8 bits), il va de soi que cet exemple n'a qu'une valeur pédagogique ; le constructeur a prévu pour réaliser des intervalles de temps longs un compteur ou *timer* dans ce microcontrôleur.

Dans notre exemple on va charger trois registres et les décrémenter jusqu'à 0.



Organigramme :

Annonce le début du programme et donne un titre à la séquense

On charge un premier registre avec d1

On charge un deuxième registre avec d2

On charge un troisième registre avec 0 ou 256

On décrémente le premier registre et on teste si son contenu = 0, si oui on continu, si non boucle pour recommencer la décrémentation.

On décrémente le deuxième registre, quand il atteint à 0 on continue la suite du programme.

On réalise les mêmes opérations pour le troisième registre.

3.8.2. *Rédaction en langage source assembleur*

3.8.2.1. *Edition du source*

```
;déclarations des variables
     CBLOCK 0x0C
     T1 : 1
     T2 : 1
     T3 : 1
     ENDC
```

```
;Equivalences => permettent de modifier
;la duree de la tempo
d1 equ 6
d2 equ 217


;sous-programme de temporisation
tempo    clrf T2          ; met T2 a 0
     movlw d1             ; charge 6 dans W
     movwf T3             ; place 6 (W) dans le registre T3
     movlw d2             ; charge 217 dans W
     movwf T1             ; met 217 dans T1
BOU  decfsz T1,f          ; decremente T1, si resultat = 0
                          ; saute l'instruction suivante
     goto BOU             ; branche a BOU
     decfsz T2,f          ; decremente et test
     goto BOU
     decfsz T3,f          ; decremente et test
     goto BOU
     return               ; fin de la temporisation et retour
                          ; éventuel
     END
```

La décrémentation jusqu'à 0 du premier registre est donc réalisée autant de fois que le contenu du deuxième multiplié par le contenu du troisième.

La décrémentation du premier registre dure :

– decfsz dure un cycle si le contenu du registre est différent de 0, et deux cycles quand le contenu du registre est nul (saut d'une instruction) ;

– goto dure 2 cycles machine et cette instruction sera réalisée T1-1 fois.

La durée totale dt est donc : (T1–1) cycles + 2 cycles + (T1–1) 2 cycles = (T1–1) 3 cycles + 2 cycles. Cette durée dt1 va être réalisée T2 fois, il faut y ajouter 2 cycles de branchement et (T2–1) 3 cycles. Cette durée dt2 sera elle aussi multipliée par T3, il faut encore y ajouter (T3–1) 3cycles et 2 cycles de branchement.

Dans l'exemple ci-dessus (T1 = 217, T2 = 256 et T3 = 6), avec un cycle machine de 1 microseconde, on obtient une temporisation totale d'environ une seconde.

L'assemblage de ce logiciel donne le listing suivant :

```
MPASM 03.10 Released            TEMPO.ASM   8-13-2002  15:32:06
PAGE  1


LOC     OBJECT  LINE          TEXT VALUE
        CODE    SOURCE
                00001  list p=16F84
                00002  include "p16F84.inc"
                00159            list
                00003
                00004 ;declarations des variables
                00005         CBLOCK 0x0C
0000    000C    00006         T1 : 1
0000    000D    00007         T2 : 1
0000    000E    00008         T3 : 1
                00009         ENDC
                00010
                00011 ;équivalences => permettent de modifier
                00012 ;la durée de la tempo
0000    0006    00013 d1 equ 6
0000    00D9    00014 d2 equ 217
                00015
0000            00016         org 0x00
0000    2801    00017         goto tempo
                00018
                00019    ;sous-programme de temporisation
0001    018D    00020 tempo   clrf T2   ; met T2 a 0
0002    3006    00021    movlw d1       ; charge 6 dans W
0003    008E    00022    movwf T3       ; place 6 (W) dans le
                                        registre T3
0004    30D9    00023    movlw d2       ; charge 217 dans W
0005    008C    00024    movwf T1       ; met 217 dans T1
0006    0B8C    00025 BOU decfsz T1,f   ; decremente T1, si résultat
                         = 0 saute l'instruction suivante
0007    2806    00026    goto BOU       ; branche a BOU
0008    0B8D    00027    decfsz T2,f    ; decremente et test
0009    2806    00028    goto BOU
000A    0B8E    00029    decfsz T3,f    ; decremente et test
000B    2806    00030    goto BOU
000C    0008    00031    return         ; fin de la temporisation et
                                        ; retour éventuel
                00032    END
```

La première colonne donne l'emplacement en mémoire
La seconde le code machine (en gras)
La troisième le numéro de ligne du listing
Le reste est la recopie du source

3.9. Fichier de codes machine « .hex » à charger en mémoire

L'assemblage du programme ci-dessus va donner un fichier de codes machines dont l'extension est ".HEX", il est rédigé en ASCII donc lisible et imprimable. Ce fichier permet au logiciel de programmation (programmateur) de placer les codes machines aux bons endroits de la mémoire de programme du microcontrôleur.

Ce fichier est composé de plusieurs enregistrements au format défini par le constructeur INTEL (INHX8M) :

```
:020000040000FA
:1000000001288D0106308E00D9308C008C0B06281B
:0A0010008D0B06288E0B0628080051
:00000001FF
```

3.9.1. *Etude de la première ligne*

« : » ce caractère désigne le début d'un enregistrement (ligne).

« 02 » : ce chiffre sur un octet donne de nombre d'octets de données contenu dans cet enregistrement.

« 0000 » : ce nombre sur 16 bits donne la valeur d'un offset (sur cette ligne il n'est pas utilisé).

« 04 » : ce chiffre désigne le type de données de la ligne. Sur cette ligne on trouve l'adresse de chargement du fichier c'est-à-dire l'adresse de début du programme. Dans ce cas cette adresse est toujours 0000h.

« 0000 » : adresse de début de programme sur 16 bits.

« FA » : est un code de contrôle d'exactitude de transmission (*checksum*).

3.9.2. *Etude de la deuxième ligne*

« : » début d'enregistrement

« 10 » : nombre d'octets de données (codes machine) : 16 octets.

« 0000 » : décalage par rapport à l'adresse de début de programme (*offset*), dans ce cas le décalage est forcément nul.

« 00 » : type de données ; sur cette ligne on trouve des codes machine.

« 01288D0106308E00D9308C008C0B0628 » : ces octets sont les codes machines au format INTEL que l'on peut retrouver sur le *listing* d'assemblage ci-dessus. Le deuxième octet est chargé à l'adresse 01h et le premier octet à l'adresse 00h.

« 1B » : *checksum*.

3.9.3. *Etude de la troisième ligne*

« : » début d'enregistrement.

« 0A » : nombre d'octets de données sur la ligne.

« 0010 » : offset, c'est-à-dire le décalage d'implantation des données qui suivent par rapport à la première adresse d'implantation. L'enregistrement précédent contient 16 octets chargés à partir de l'adresse 0000h, les données de cette ligne sont donc décalées de 16 octets (10h).

« 00 » : les octets de cette ligne sont des codes machines.

« 8D0B06288E0B06280800 » : codes machines à implanter.

« 51 » : *checksum.*

3.9.4. *Etude de la dernière ligne*

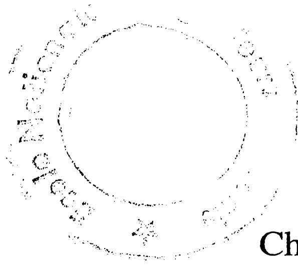
« : » début d'enregistrement

« 00 » : longueur de l'enregistrement ; il n'y a pas de donnée.

« 0000 » : *offset* nul.

« 01 » : type de données ; fin de fichier.

« FF » : *checksum.*



Chapitre 4
Le jeu d'instructions

4.1. Format des instructions

Une instruction est codée en un seul mot de 14 bits.

Ce mot de 14 bits est composé :

– d'un code opération ou code machine (OPCODE), disposé sur les bits de poids forts du mot, précisant le type d'instruction ;

– d'une ou plusieurs opérande(s) codée(s) sur les bits restants, précisant l'opération de l'instruction.

Les instructions permettent de manipuler des données sur 8 bits, on a donc affaire à un microcontrôleur 8 bits exclusivement.

On distingue quatre grands types d'instruction :

– les instructions orientées « mot ». Le traitement porte simultanément sur 8 bits ;

– les instructions orientées « bit ». Le traitement porte sur un seul bit désigné ;

– les instructions de contrôle ou de branchement.

Le microcontrôleur PIC est dit à architecture RISC (Reduce Instruction Set Computeur) c'est à dire à jeu d'instructions réduit, il ne dispose en effet que de 35 instructions distinctes.

4.1.1. *Opérations orientées « mots » des registres* « FILE »

Le registre W (*Working register*) est l'unique accumulateur du processeur.

Une instruction de calcul (addition, soustraction, ET logique..etc.) sollicite l'ALU qui travaille uniquement sur W et un registre de la zone RAM « FILE ». Dans

ce cas des instructions orientées « mot », selon l'état du bit « d » du rang 7 de l'instruction, le résultat de l'instruction sera logé dans W si d = 0 ou dans le registre considéré de la zone « FILE » si d = 1.

f est l'adresse d'un registre « FILE » sur 7 bits.

b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
code opération					d	f(FILE)						

EXEMPLE.–

Soit l'instruction ADDWF 0x40,1

Code correspondant : 000111 code machine	d choix du lieu de stockage	fff ffff . adresse du registre

Le registre 0x40 se trouve dans la zone « FILE » (RAM).

Cette instruction accomplit l'addition : W + (0x40) —> (0x40)

Son codage sur 14 bits est le suivant :

0 0 0 1 1 1	1	1 0 0 0 0 0 0
ADDWF	d = 1	0x40

Le résultat est placé dans le registre considéré de la zone « FILE » car d = 1.

Pour loger le résultat du calcul dans W, l'écriture de l'instruction est :
ADDWF 0x40,w   et son codage :
0 0 0 1 1 1   0   1 0 0 0 0 0 0

Dans ce cas d = 0.

Nota : le fichier d'équivalence P16F84.INC permet à l'assembleur de définir la valeur du bit d ; il donne w equ 0 et f equ 1. La syntaxe de l'instruction est simplifiée pour le programmeur, il lui suffit d'écrire ADDWF 0x40,f ou ADDWF 0x40,w.

4.1.2. *Opérations de manipulations de bits*

b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
code opération			adresse bit			f						

L'adresse du bit est écrite sur 3 bits et f est l'adresse du registre sur 7 bits dans la zone « FILE ».

EXEMPLE.–

BCF (Bit Clear F) met à 0 un bit dans un registre f.

le code correspondant : 0100	bbb	fffffff
code machine	adresse du bit	adresse du registre

4.1.3. *Opérations de branchement*

b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
code opération			K (valeur immédiate sur 11 bits)									

L'adresse de destination sur 11 bits ne permet qu'un saut à l'intérieur d'une page de mémoire (2 Ko). Pour accèder à une adresse plus lointaine il faut alors mettre en œuvre les bits b3 et b4 du registre PCLATH.

EXEMPLE.– GOTO toto

Toto est l'étiquette correspondant à la destination

Le code correspondant : 101	xxxxxxxxxxx
code machine	adresse de destination

4.2. Les instructions du PIC16F84

Les instructions sont classées dans le tableau suivant.

On trouve de la gauche vers la droite :
- le mnémonique ;
- la fonction réalisée ;
- les modifications du registre d'état résultant de l'opération.

4.2.1. *« ADDLW » (ADD Literal and W)*

Cette opération permet d'ajouter une valeur « literal » (adressage immédiat) au contenu du registre de travail W.

Syntaxe : addlw k ; (W) + k -> (W)

Mnémonique	Instructions sur les registres (octets)	Bits modifiés
ADDWF f,d	d:=w+f ; ajoute w au contenu de f	C,DC,Z
ANDWF f,d	d:=w AND f ; ET entre w et f	Z
CLRF f	f:=0 ; met à 0 f	Z
CLRW	w:=0 ; met à 0 w	Z
COMF f,d	d:=NOT(f) ; complémente f	Z
DECF f,d	d:=f -1 ; décrémente f	Z
DECFSZ f,d	d:=f -1 ; décrémente f et saut si le résultat = 0	-
INCF f,d	d:=f+1 ; incrémente f	Z
INCFSZ f,d	d:=f+1 ; incrémente f et saut si le résultat = 0	-
IORWF f,d	d:=w OR f ; OU entre w et f résultat dans d	Z
MOVF f,d	d:=f ; charge le contenu de f dans d	z
MOVWF f	f:=w ; sauve w dans f (direct)	-
NOP	n'effectue aucune opération	-
RLF f,d	d=f SHL 1 ; rotation à gauche avec la retenue	C
RRF f,d	d=f SHR 1 ; rotation à droite avec la retenue	C
SUBWF f,d	d:= f-w ; soustrait W de f (direct)	C,DC,Z
SWAPF f,d	d:= f[4..7] <-- > f[0..3] ; inverse les quartets	-
XORWF f,d	d:= w XOR f ; OU exclusif entre w et f.	Z
	Instructions sur les registres (bit par bit)	
BCF f,b	f[b]:=0 ; mets à 0 le bit b de f	-
BSF f,b	f[b]:=1 ; mets à 1 le bit b de f	-
BTFSC f,b	teste le bit b de f ; saut si le bit est à 0	-
BTFSS f,b	teste le bit b de f ; saut si le bit est à 1	-
	Instructions de contrôle et branchement	
ADDLW k	w:=w+k ; ajoute k à w	C,DC,Z
ANDLw k	w:=w AND k ; ET entre w et k	Z
CALL k	appel un sous programme	-
CLRwDT	remet à 0 le compteur du chien de garde	TO,PD
GOTO k	se branche à l'adresse k	-
IORLw k	w:=w OR k ; Ou entre w et k	Z
MOVLw k	w:=k ; charge k dans w	-
RETFIE	retour d' interruption	-
RETLw k	w:=k , puis effectue un retour de sous programme	-
RETURN	effectue un retour de sous programme	-
SLEEP	place le circuit en mode sommeil et stoppe l'oscillateur	TO,PD
SUBLw k	w:=k-w ; soustrait w de k	C,DC,Z
XORLw k	w:=w X OR k ; OU exclusif entre w et k	Z

f : adresse d'un registre sur 7 bits
k : valeur immédiate sur 8 bits (0 à 127)
si d = 0 destination W (registre W)
si d = 1 destination f (zone file)

Z : Zéro
TO : Time Out
PD : Power Down
C : Carry (retenue)
DC : Demi Carry

Bits du registre STATUS affectés :

– Z Si le résultat de l'opération est 0, Z passe à 1 ;

– C Si le résultat de l'opération est supérieur à 0xFF (255), C passe à 1 ;

– DC Si le résultat de l'opération entraîne un report du bit 3 vers le bit 4, DC vaudra 1. DC n'est utilisé que pour les opérations sur les quartets, par exemple, les nombres BCD (*Binary Coded Decimal*).

Exemple :

```
movlw 253       ; charger 253 en décimal dans W
addlw 3         ; Ajouter 3 dans W
                ; W contient 0 (256 ou h100), Z vaut 1,
                ; C vaut 1(débordement)
```

4.2.2. « *ADDWF* » *(ADD W and F)*

Le contenu du registre W est ajouté au contenu du registre f. Il s'agit ici d'un adressage direct.

Syntaxe : addwf f , d ; (w) + (f) -> (d)

Bits du registre STATUS affectés : C, DC, et Z.

Exemple :

```
movlw  5        ; charger 5 dans W
movwf toto      ; toto vaut maintenant 5
movlw 25        ; charger 25 dans W
addwf toto,1    ; résultat : (W) + (x)
                ; résultat = 30 sauvé dans toto car d=1.
```

4.2.3. « *ANDLW* » *(AND Literal with W)*

Cette instruction effectue une opération « ET » logique entre le contenu de W et la valeur « literal » qui suit (adressage immédiat), le résultat est dans W.

Syntaxe : andlw k ; (w) & k -> (w) avec k = octet

Bit du registre STATUS affecté : Z

Exemple :

```
movlw B'11001101'      ; charger w
andlw B'11110000'      ; effectuer un 'ET logique'
                       ; résultat : B'11000000' dans W
```

Cette instruction peut servir à la réalisation d'un masque utilisé pour sélectionner des bits dans un mot.

4.2.4. *« ANDWF » (AND W with F)*

Syntaxe Andwf f , d ;(f) AND (w) -> (d)

Réalise un masque entre le contenu de w et le contenu de f (adressage direct) ; le résultat est stocké en fonction de d.

Bit du registre STATUS affecté : Z.

Exemple :

Éliminer les bits de poids faible dans un registre f

Avant dans f : 45h

```
movlw 0xF0          ; charger le masque 0x0F dans W
andwf x,1           ; on élimine le quartet de poids
                    ; faible, le résultat est dans f.
```

Après dans f : 40h.

4.2.5. *« BCF » (Bit Clear F)*

Permet de mettre à 0 un bit d'un registre.

Syntaxe : bcf f , b ; le bit défini par b (sur 3 bits) est mis à 0 dans le registre (f) , b est compris entre 0 et 7

Bit du registre STATUS affecté : aucun.

Exemple :

Avant : (x) = B'1111 1111'

```
bcf x,5   ; met le bit 5 de (x) à 0
```

Après : (x) = B'1101 1111'

4.2.6. *« BSF » (Bit Set F)*

Permet de forcer à 1 un bit d'un registre.

Syntaxe : bsf f,b ; b donne la position du bit dans la case mémoire (f)
; b est compris entre 0 et 7

Bit du registre STATUS affecté : aucun.

Exemple :

Avant : (x) = B'0000 0000'

```
bsf x,2   ; positionne le bit 2 de (x) à 1
```

Après : (x) = B'0000 0100'

4.2.7. « *BTFSC* » *(Bit Test F, Skip if Clear)*

Teste le bit sélectionné du registre et saute l'instruction suivante si le bit vaut 0. Dans ce cas l'instruction prend 2 cycles, sinon, elle n'utilise qu'un cycle machine.

Syntaxe : btfsc f, b ; teste le bit repéré par b de la case
; mémoire (f).

Bit du registre STATUS affecté : aucun.

Exemple :
```
ici BTFSC toto,1
    non goto XXX
    oui
```

Avant : PC = étiquette ici,

Après : si le bit 1 de TOTO = 0, PC prend la valeur étiquette oui et si le bit =1 PC prend la valeur étiquette non.

4.2.8. « *BTFSS* » *(Bit Test F, Skip if Set)*

Teste le bit du registre et saute s'il vaut 1.

Syntaxe : btfsc f, b ; on teste le bit b de la mémoire (f).
; si ce bit vaut 1, on saute l'instruction suivante
; sinon on exécute l'instruction suivante.

Bit du registre STATUS affecté : aucun

Exemple :

```
btfss STATUS,C      ; tester si le bit C du registre
                    ; STATUS vaut 1
```

4.2.9. « *CALL* » *(CALL sous-programme)*

Syntaxe : call etiquette ; appel du sous-programme à l'adresse étiquette.

Mécanisme : lors de l'exécution de l'instruction CALL, l'adresse de l'instruction suivante est sauvegardée dans la pile (c'est l'adresse de retour). Lorsque le sous-programme est terminé, l'adresse sauvegardée est retirée de la pile et placée dans le PC. Le programme poursuit alors depuis l'endroit d'où il était parti. La PILE ne disposant que de 8 emplacements, il faut limiter le nombre d'imbrications de sous-programmes.

Bit du registre STATUS affecté : aucun

4.2.10. « *CLRF* » *(CLeaR F)*

Cette instruction efface le registre f spécifié

Syntaxe : clrf f ; (f) = 0

Bit du registre STATUS affecté : Z vaut donc toujours 1 après cette opération.

Exemple : `clrf PORTA ; (PORTA) = 0`

4.2.11. « *CLRW* » *(CLeaR W)*

Cette instruction efface le contenu de l'accumulateur W (met à zéro).

Syntaxe : clrw ; (w) = 0

Bit du registre STATUS affecté : Z vaut toujours 1 après cette opération.

On pourrait utiliser l'instruction « movlw 0 », mais dans ce cas le bit Z n'est pas affecté.

4.2.12. « *CLRWDT* » *(CLeaR WatchDog)*

Remet à 0 le compteur du chien de garde, opération qui permet de provoquer un *reset* automatique du microcontrôleur en cas d'arrêt du programme provoqué, par exemple, par un parasite.

Le programme doit envoyer cette instruction à intervalles réguliers. Si cette commande n'est pas reçue dans le délai imparti, le microcontrôleur est redémarré.

Syntaxe : clrwdt ; remet le compteur du watchdog à 0

Bits du registre STATUS affectés : TO est mis à 1 et PD est mis à 1.

4.2.13. « *COMF* » *(COMplement F)*

Effectue le complément à 1 du registre spécifié.

Syntaxe : comf f , d ; NOT (f) -> (d)

Bit du registre STATUS affecté : Z

Exemple :

```
movlw B'11001010'   ; charge une valeur dans W
movwf x             ; met cette valeur dans x
comf x,w            ; met l'inverse de x dans W
                    ; (W) = B'00110101'
```

4.2.14. « *DECF* » *(DECRement File)*

Decrémente le registre spécifié et le met en mémoire selon la valeur de d.

Syntaxe Decf f , d ; (f) – 1 -> (d)

Bit du registre STATUS affecté : Z.

Si avant l'instruction, (f) vaut 1, Z vaudra 1 après l'exécution (1-1 = 0)

Exemple :

```
decf x , f     ; décrémente x, résultat dans x
decf x , w     ; prends (x)—1 et place le résultat dans w.
```

4.2.15. *« DECFSZ » (DECrement F, Skip if Z)*

Décrémente un registre f et saute l'instruction suivante si le résultat de la décrémentation donne une valeur nulle. Si le résultat est nul, l'équivalent d'un NOP est exécuté (durée : 2 cycles).

Syntaxe : decfsz f, d ; (f) –1 -> (d).

Bit du registre STATUS affecté : aucun

Exemple :

```
      movlw 30              ; charger 30 dans W
      movwf compteur        ; initialiser compteur
boucle decfsz compteur,1    ; décrémenter compteur et tester sa
                            ; valeur résultat dans compteur.
      goto boucle           ; tant que le compteur n'est pas = à
                            ; 0, on boucle
```

4.2.16. *« GOTO » (aller à)*

Cette instruction effectue ce que l'on appelle un saut inconditionnel (équivalent de *jump*).

L'instruction goto contient les 11 bits de l'emplacement de destination. Les 2 bits manquants pour reconstituer l'adresse complète sont dans le registre PCLATH.

Fonctionnement de goto :

- l'adresse de saut sur 11 bits est chargée dans le PC ;
- les 2 bits manquants sont chargés depuis PCLATH (b3 et b4) ;
- le résultat donne l'adresse sur 13 bits ;
- la suite du programme s'effectue à la nouvelle adresse du PC.

L'adresse de saut = adresse réelle.

Syntaxe : goto etiquette

nota : goto nécessite 2 cycles d'horloge.

4.2.17. *« INCF » (INCrement File)*

Cette instruction provoque l'incrémentation du registre f.

Syntaxe : incf f,d

d vaut toujours, au choix :

– f (la lettre f) : le résultat est stocké dans l'emplacement mémoire ;

– w (la lettre w) : le résultat est placé dans l'accumulateur, et le contenu de l'emplacement mémoire n'est pas modifié.

Bit du registre STATUS affecté : Z.

Exemples :

```
incf x , f      ; le contenu de x est augmenté de 1
                ; le résultat est stocké dans x.
incf x , w      ; le contenu de X est chargé dans W et
                ; augmenté de 1. W contient donc le contenu de
                ; x+1. x n'est pas modifié
```

4.2.18. *« INCFSZ » (INCrement F, saut si Zero)*

Cette instruction incrémente un registre et fait un saut au-dessus de l'instruction suivante (goto Bou) si le contenu du registre atteint 0.

Syntaxe : incfsz f , d ; (f) + 1 -> (d) si (d) = 0 le résultat est placé ; dans W

goto BOU ; si le résultat est 1 l'instruction suivante est ; exécutée (goto)

suite ======= ; si le résultat est nul un NOP est exécuté (suite)

Bit du registre STATUS affecté : aucun

Exemple :

```
Bou incfsz compteur,f   ; si le résultat dans compteur est 0
    goto bou            ; l'instruction située à bou2 est exécutée.
bou2 =====
```

4.2.19. *« IORLW » (Inclusive OR Literal with W)*

Syntaxe : iorlw k ; (w) OR k -> (w)

Réalise une fonction OU logique entre le contenu de W et l'opérande K. Le résultat est mis dans W.

Bit du registre STATUS affecté : Z

Exemple :

```
movlw 0x9A      ; charger 0x9A dans W
iorlw 0x35      ; résultat : (w) = 0xBF et Z = 1
```

4.2.20. *« IORWF » (Inclusive OR W avec File)*

Effectue un OU entre (W) et le contenu du registre (adressage direct), le résultat est orienté en fonction de la valeur de d.

Syntaxe : Iorwf f , d ; (w) OR (f) -> (d)

Bit du registre STATUS affecté : Z

Exemple : avant : PORTA contient 0x01 et w contient 0x03

```
iorwf PORTA,1
```

après : PORTA contient 0x02

4.2.21. *« MOVF » (MOVe File)*

Charge le contenu du registre spécifié dans la destination selon la valeur de d.

Syntaxe : movf f,d ; (f) -> (d)

Bit du registre STATUS affecté : Z (si f vaut 0, Z vaut 1).

Exemple :

```
movf x,w        ; met le contenu de x dans w.
```

4.2.22. *« MOVLW » (MOVe Literal to W)*

Cette instruction charge la valeur spécifiée (valeur « literal » ou encore valeur immédiate) dans l'accumulateur W.

Syntaxe : movlw k ; k est une valeur immédiate de $00 à $FF.

Bit du registre STATUS affecté : aucun

Exemple :

```
movlw 0x25      ; charge 25h dans le registre W
```

4.2.23. *« Movwf » (MOVe W to File)*

Transfert le contenu du registre de travail W dans un registre « FILE ».

Syntaxe movwf f ; (W) -> (f)

Aucun bit du registre STATUS n'est affecté.

Exemple :

```
movlw 0x50      ; charge 0x50 dans W
movwf PORTB     ; PORTB contient maintenant 0x50.
```

4.2.24. *« NOP » (No Operation)*

Instruction qui ne fait rien et ne modifie rien mais occupe du temps.

Syntaxe : nop

4.2.25. « RETFIE » (RETurn From IntErrupt)

Cette instruction provoque un retour d'interruption. Cette instruction agit d'une manière identique à RETURN, excepté que les interruptions sont remises automatiquement en service au moment du retour au programme principal car les masques sont restés à 1, (le programmeur doit les remettre à 0).

Syntaxe : retfie ; retour d'interruption

Bit du registre STATUS affecté : aucun

4.2.26. *« RETLW » (RETurn with Literal in W)*

Retour de sous-programme avec valeur dans W. C'est une instruction qui équivaut à l'instruction *return*, mais permet de sortir d'un sous-programme avec une valeur spécifiée dans W.

Syntaxe : retlw k ; (w) = k puis retour

Bit du registre STATUS affecté : aucun

Exemple :

```
test            ; étiquette de début de notre sous-programme
btfss PORTB,3   ; teste le bit 3 de PORTB et saut éventuel
retlw 0         ; si le bit vaut 0, fin de sous-programme avec
                ; l'accumulateur W contenant 0
retlw 1         ; sinon, retour avec W contenant 1
```

4.2.27. *« RETURN » (RETURN from subroutine)*

Retour de sous-programme. Cette instruction est indispensable à la fin d'un sous-programme, elle permet de recharger le compteur de programme avec l'adresse de retour qui a été sauvegardée dans la pile (adresse de l'instruction suivant CALL).

Syntaxe : Return ; fin de sous-programme.

Bit du registre STATUS affecté : aucun

Exemple : réalisation d'une temporisation.

Dans l'exemple qui suit le programme principal fait appel à un sous-programme grâce à l'instruction CALL. Celui-ci se termine avec l'instruction RETURN.

Le sous-programme destiné à réaliser une temporisation effectue une opération de décomptage qui prend du temps. La durée de ce décomptage est donnée par un paramètre (une valeur) qui fixe le nombre de décrémentations ; c'est le nombre de boucles qui fait la durée. Ce paramètre est fixé dans le programme principal en chargeant le l'accumulateur W, on dit qu'il y a passage de paramètres.

```
;Programme principal :

    ======          ; instruction quelconque
    movlw 0x25      ; charger W avec 0x25
    call tempo      ; appel du sous programme tempo d'une
                    ; durée de 0x25
    ======          ; instruction quelconque
    movlw 0x50      ; charger W avec 0x50
    call tempo      ; appel du sous programme tempo d'une
                    ; durée de 0x50
    xxx             ; instruction quelconque
```

```
;Sous-programme avec passage de paramètre.

tempo movwf compteur        ; initialiser le compteur de boucles
                            ; avec le contenu de w (le paramètre).
boucle decfsz compteur,f    ; décrémenter compteur, test, sauter
                            ; si résultat = 0
     goto boucle            ; boucler si résultat pas 0
     return                 ; fin de la sous-routine et retour au
                            ; programme principal.
```

4.2.28. *« RLF » (Rotate Left through Carry)*

Rotation vers la gauche en utilisant la retenue. Le bit de retenue C du registre d'état est décalé dans le registre f et le résultat est placé en fonction de la valeur de d.

Syntaxe : rlf f , d ; (f) rotation gauche avec C -> (d)

Bit du registre STATUS affecté : C

Exemple :

```
bsf STATUS,C          ; positionne le carry à 1
movlw B'00010111'     ; charge une valeur dans w
movwf X               ; met w dans X
rlf x,f               ; rotation vers la gauche avec la
                      ; retenue C
```

Avant : X = 0001 0111

Après : X = 0010 1111

4.2.29. *« RRF » (Rotate Right through Carry)*

Rotation vers la droite en utilisant le bit C de retenue.

Syntaxe : rrf f , d ; (f) rotation droite avec carry-> (d)

Bit du registre STATUS affecté : C

Exemple :

```
bsf STATUS,C        ; positionne le carry à 1
movlw B'00010000'   ; charge une valeur dans w
movwf X             ; w dans X
rrf x,f             ; rotation vers la droite
```

Avant : X = 0001 0000

Après : X = 1000 1000

4.2.30. *« SLEEP » (mise en sommeil)*

Place le microcontrôleur en sommeil.

Syntaxe : Sleep ; arrêt de l'horloge

Bits du registre STATUS affectés : PD est mis à 0 et TO est mis à 0.

4.2.31. *« SUBLW » (SUBtract W from Literal)*

On soustrait en complément à 2 le contenu de W à la valeur littérale (opérande) et le résultat est placé dans W (adressage immédiat).

Syntaxe : sublw k ; k – (W) -> (W)

Bits du registre STATUS affectés : C, DC, Z

Exemple :

```
movlw 0x01          ; charger 0x01 dans W
sublw 0x02          ; soustraire W de 2
                    ; résultat : 2 – (W) = 2 - 1 = 1
                    ; Z = 0, C = 1, donc résultat positif
```

Exemple :

```
movlw 0x02          ; charger 0x02 dans W
sublw 0x02          ; soustraire 2 – (W) = 2 – 2 = 0
                    ; Z = 1 , C = 1 : résultat nul
```

Exemple :

```
movlw 0x03          ; charger 0x03 dans W
sublw 0x02          ; soustraire 2 – (W) = 2 – 3 = -1
                    ; C = 0, Z = 0 : résultat négatif.
```

4.2.32. *« SUBWF » (SUBtract W from F)*

Soustraction du contenu de W à un registre f, mode d'adressage direct.

Syntaxe : subwf f,d ; (f) – (W) -> (d)

Bits du registre STATUS affectés : C , DC , Z

Exemple :

Si le contenu de W = 2

```
SUBWF reg,1
```

Si reg = 3 ; le résultat (2) est mis en mémoire dans le registre, C = 1 et Z = 0

Si reg = 2 ; Z = 1 et C = 1 car le résultat est nul.

Si reg = 1 ; Z = 0 et C = 0 car le résultat est négatif.

4.2.33. « *SWAPF* » *(SWAP nibbles in F)*

Cette opération inverse le quartet de poids faible avec celui de poids fort dans le registre spécifié.

Syntaxe : swapf f , d ; inversion des b0/b3 de (f) avec b4/b7 -> (d)

Bit du registre STATUS affecté : aucun

Exemple :

```
movlw 0x65          ; charger 0x65 dans W
movwf x             ; placer dans x
swapf x,f           ; (x) = 0x56, résultat dans f.
```

4.2.34. « *XORLW* » *(eXclusive OR Literal with W)*

Syntaxe : xorlw k ; (W) xor k -> (W)

Réalise un Ou exclusif entre W et l'opérande.

Bit du registre STATUS affecté : Z

Exemple :

```
movlw B'10110101'   ; charger W
xorlw B'10101111'   ; résultat : B '00011010' ;
```

4.2.35. « *XORWF* » *(eXclusive OR W with F)*

Même opération que XORLW, mais en adressage direct.

Syntaxe : xorwf f , d ; (w) xor (f) -> (d)

Bit du registre STATUS affecté : Z

Chapitre 5
Le microcontrôleur

5.1. Rôle d'un microcontrôleur

Le microprocesseur que l'on appelle unité centrale, car ce n'est qu'une des composantes d'un microcontrôleur, n'a été jusqu'à présent utilisé que pour des traitements d'informations :

– transfert entre registre et mémoire ;

– calcul ou transformation de données.

Il doit aussi pouvoir recevoir et transmettre des informations *via* des périphériques. Pour cela on trouvera entre l'unité centrale et les périphériques, des interfaces pour adapter les caractéristiques du microprocesseur aux organes externes.

Adaptation en temps

Le microprocesseur travaille à des vitesses élevées généralement supérieures à un périphérique, parfois composé d'organes mécaniques ou électromécaniques.

Adaptation de niveau ou de logique

Il est souvent nécessaire d'adapter les signaux issus des capteurs d'informations.

Adaptation de format des données

L'accès aux périphériques peut se faire avec des données parallèles ou des données séries. Une conversion peut donc être nécessaire. Les données peuvent aussi être analogiques d'où un besoin de conversion analogique/digitale ou digitale/analogique. Chacune de ces adaptations est réalisée par une interface spécifique.

Celle-ci contient généralement :

– un ou plusieurs registres de contrôle permettant de définir son fonctionnement ;

– un registre d'état donnant des indications sur le déroulement des opérations ;

– un ou plusieurs registres permettant le transfert des données.

Chaque registre peut être considéré comme une case mémoire avec une adresse et sera donc accessible par l'unité centrale comme telle.

Le microcontrôleur est donc un circuit comprenant à la fois une unité centrale ou microprocesseur et une ou plusieurs interfaces.

5.2. Le microcontrôleur PIC16F84

Le choix du modèle dépend principalement de l'utilisation que l'on veut en faire. Il en existe de très nombreuses versions. Notre but étant d'apprendre à l'utiliser, nous avons choisi un modèle simple et pourtant bien approprié pour la mise au point de prototypes de petites applications : le PIC16F84.

Le lecteur n'aura aucune difficulté à travailler sur un autre modèle car les modes de fonctionnement sont identiques, seules changent certaines interfaces.

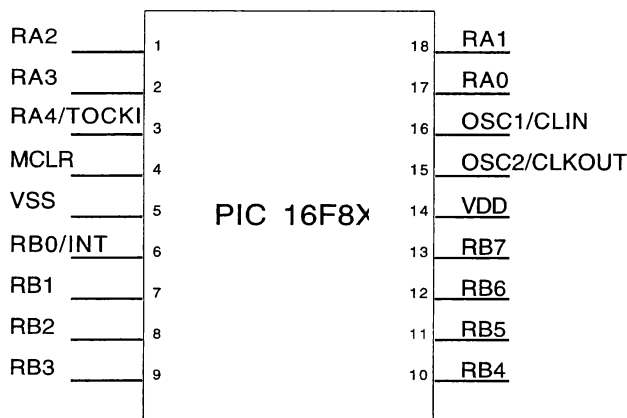


Figure 5.1. *Le PIC16F84*

Dans le cas du PIC16F84, on distingue :

– la mémoire programme composée de 1024 mots de 14 bits d'EEPROM FLASH ;

– 68 octets de RAM à usage général et 22 registres spécifiques ;

– 64 octets de EEPROM pour des données de configuration ou d'exploitation (consignes...) ;

– un compteur/timer et un chien de garde ;

– un port A de 5 lignes (RA0 à RA3) et un port B de 8 lignes (RB0 à RB7).

5.3. Organisation de la mémoire

La mémoire comprend 2 zones distinctes ; la zone donnée et la zone programme possédant chacune leur propre bus et une zone de mémoire EEPROM.

5.3.1. *Les registres de travail*

La zone de données (registres de travail) est constituée de 256 octets de mémoire RAM, elle est nommée par le constructeur « FILE » comme un fichier.

	Bloc mémoire 0	Bloc mémoire 1	
$00	Reg. adressage indirect	Reg. adressage indirect	$80
$01	TMRO	option	$81
$02	pcl	pcl	$82
$03	status	status	$83
$04	fsr	fsr	$84
$05	porta	trisa	$85
$06	portb	trisb	$86
$07			$87
$08	eedata	eecon1	$88
$09	eesadr	eecon2	$89
$0A	pclath	pclath	$8A
$0B	intcon	intcon	$8B
$0C $4F	*68 registres en RAM à usage général*	*Cette zone est adressée en même temps que la zone mémoire 0.*	$8C $CF
$50 - - $7F	Zone non utilisable : donne 0 en lecture		$D0 - - $FF

Figure 5.2. *L'organisation des registres de travail*

Elle est divisée en 2 blocs mémoire sélectionnables par le bit RP0 du registre STATUS. Avant d'accéder à certains registres, il faut s'assurer que le bloc mémoire correspondant est sélectionné.

Cette sélection se fait par logiciel; si RP0 = 0 on accède au bloc 0.

Cette zone de données s'étale de 00h à FFh et comprend deux zones distinctes :

– les registres à usage spécial (*Special Fonction Registers*) comprenant le registre d'état du noyau du processeur (CORE) et les registres de contrôle et d'état des périphériques (en haut de la carte mémoire) ;

– les registres à usage général (*General Purpose Registers*) soit 68 octets de RAM.

REMARQUE.– dans le PIC16F877, il y a quatre blocs de mémoire ce qui justifie l'existence des bits RP0 et RP1 dans le registre STATUS

5.3.2. *La zone programme*

Seuls les 1 024 premiers mots sont utilisés et matérialisés par de la mémoire EEPROM FLASH que l'on peut programmer et effacer électriquement environ 1 000 fois.

Le logiciel étudié est testé à l'aide d'un simulateur puis est chargé dans la mémoire à l'aide d'un programmateur, cette procédure est lourde et la durée de vie d'un tel microcontrôleur est assez réduite. Toutefois elle est valable aussi bien pour la mise au point du logiciel que dans son utilisation définitive.

L'espace programme s'étale de $0000 à $03FF, c'est dans cette zone que le programme est écrit. Le PC (compteur programme) contient 13 bits qui permettent d'adresser jusqu'à 8K mots de 14 bits (taille d'une instruction).

Le vecteur RESET se trouve en 0000h.

L'unique vecteur lié aux autres interruptions se trouve en 0004h, (un registre de contrôle et d'état des interruptions permet de connaître la source de l'interruption survenue (port, *timer* ...).

5.3.3. *La mémoire EEPROM*

Composée de 64 octets de mémoire EEPROM à la disposition de programmeur, elle permet de stocker des informations qui ne seront pas effacées par une coupure de l'alimentation. On y accède par une procédure spéciale.

5.3.4. *Fonctions réalisées par le microcontrôleur*

– transfert unidirectionnel ou bidirectionnel de données en parallèle avec les ports A et B ;

– temporisation/comptage permettant de :
- mesurer des intervalles de temps,
- compter des impulsions,
- générer des interruptions en temps réel (horloge temps réel),
- surveiller le bon déroulement d'un programme (*watchdog* ou chien de garde).

Certaines lignes d'entrées/sorties sont polyvalentes, elles peuvent être utilisées :

– comme des entrées/sorties parallèles uni ou bidirectionnelles (ports A et B) ;

– comme des entrées de demande d'interruption, ou de comptage.

5.4. Description des connexions

VDD et VSS : ce sont les connexions d'alimentation du circuit. Le circuit est réalisé en technologie HCMOS, il est alimenté avec une tension de 5 volts ce qui le rend compatible avec les circuits CMOS et TTL. Il est conseillé de bien découpler cette connexion d'alimentation (commutation rapide).

MCLR : cette connexion, active au niveau 0, est l'entrée de RESET (*Master CLear Reset*) elle permet aussi le branchement de la haute tension Vpp nécessaire à la programmation du composant.

RA0 à RA4 : constituent le PORTA du microcontrôleur. Ce port bidirectionnel est compatible TTL. La ligne RA4 de type drain ouvert en sortie peut aussi être utilisée comme entrée d'horloge du compteur-timer TMR0. En entrée cette ligne est de type « bascule de Schmitt ».

RB0 à RB7 : constituent le PORTB du microcontrôleur. Ce port bidirectionnel est compatible TTL. La ligne RB0 peut servir d'entrée de demande d'interruption externe (INT), dans ce cas elle est de type « bascule de Schmitt ».

Les lignes RB4 à RB7 peuvent servir d'entrées d'interruption, dans ce cas elles déclenchent une demande d'interruption à chaque fois que l'une d'elles change d'état.

Les entrées RB6 et RB7 de type « bascule de Schmitt » permettent aussi la programmation en mode série du microcontrôleur. Dans ce cas la ligne RB6 est l'entrée d'horloge et la ligne RB7 est l'entrée de données.

OSC1 et OSC2 : ces connexions permettent l'entrée des signaux nécessaires au fonctionnement de l'horloge de l'unité centrale.

Le microcontrôleur PIC peut accueillir quatre types d'horloge, chaque type étant sélectionné grâce aux bits FOSC1 et FOSC2 du registre CONFIG.

Nom	N° DIP	N° SOIC	Type I/O	Type	Description
OSC1/CLKIN	16	16	I	ST/CMOS[3]	Entrée du quartz ou entrée d'horloge externe
OSC2/CLKOUT	15	15	O	—	Connexion du quartz ou sortie d'oscillateur
MCLR	4	4	I/P	ST	Entrée de reset active au niveau 0 ou entrée de tension de programmation.
					Le port A est bidirectionnel
RA0	17	17	I/O	TTL	
RA1	18	18	I/O	TTL	
RA2	1	1	I/O	TTL	
RA3	2	2	I/O	TTL	
RA4/TOCKI d'horloge	3	3	I/O	ST	Sortie en drain ouvert. Peut aussi être utilisé comme entrée du TIMER0.
					Le port B est bidirectionnel. On peut par programme lui affecter des résistances de rappel de potentiel
RBO/INT	6	6	I/O	TTL/ST[1]	Peut aussi être utilisée comme entrée d'interruption.
RB1	7	7	I/O	TTL	
RB2	8	8	I/O	TTL	
RB3	9	9	I/O	TTL	
RB4	10	10	I/O	TTL	Détection d'interruption.
RB5	11	11	I/O	TTL	Détection d'interruption.
RB6	12	12	I/O	TTL/ST[2]	Détection d'interruption ou entrée d'horloge de programmation.
RB7	13	13	I/O	TTL/ST[2]	Détection d'interruption ou entrée de données de programmation
Vss	5	5	P	—	Masse
VDD	14	14	P	—	Alimentation

I = entrée, O = sortie, I/O = entrée / sortie, TTL = norme TTL, ST = entrée avec trigger de Schmitt.
1 : en entrée d'interruption avec trigger de Schmitt. 2 : avec trigger de Schmitt en mode programmation. 3 : avec trigger de Schmitt quand le microcontrôleur utilise un oscillateur RC, sinon entrée CMOS.

Fosc1	Fosc0	Type d'oscillateur
0	0	Oscillateur avec réseau résistance capacité. Fmax : 4 MHz.
0	1	Oscillateur XT à quartz ou oscillateur extérieur. Fmax : 4 MHz
1	0	Oscillateur HS à quartz grande vitesse. Fmax : 10MHz.
1	1	Oscillateur LP à quartz basse consommation. Fmax : 200KHz

Figure 5.3. *Mode XT-HS*

Figure 5.4. *Mode RC*

En mode XT, LP et HS (figure 5.3) un quartz ou un résonnateur (filtre céramique) est branché entre les connexions OSC1 et OSC2.

En cas de programmation du mode RC (figure 5.4), il ne faut jamais utiliser OSC1 comme entrée d'horloge sous peine de destruction du composant. La connexion OSC2 délivre dans ce cas une fréquence divisée par 4 et peut être utilisée pour synchroniser d'autres composants.

Figure 5.5. *Mode extérieur*

En cas d'utilisation d'un oscillateur externe (figure 5.5) branché sur la connexion OSC1, la connexion OSC2 est en sortie ouverte.

5.5. Le registre OPTION

OPTION_REG	RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0
81h	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1

Le registre OPTION est un registre utilisable en lecture et en écriture. Il permet de configurer le prédiviseur du compteur, l'interruption externe INT et de mettre en place des rappels de potentiel sur le port B. Après le RESET tous ses bits sont à 1.

RBPU : Pull up du port B.

Une résistance de rappel (environ 50 KΩ) est relié au +5 volt sur chaque connexion du PORTB, quand le bit RBPU est mis à 0 (actif au niveau bas).

REMARQUE.– Ce bit valide simultanément les résistances sur toutes les connexions du PORTB.

INTEDG : sélection du front du signal de demande d'interruption.

Si INTEDG = 1, une interruption est demandée si le niveau sur la connexion RB0 (INT) passe de 0 vers 1 (front montant).

Si INTEDG = 0, l'interruption s'effectuera lors de la transition de 1 vers 0 (front descendant).

TOCS : sélection de la source d'horloge du compteur.

Le timer0 est incrémenté par l'horloge interne Fosc/4 (synchronisé au programme) si TOCS = 0, sinon il compte les impulsions reçues sur la connexion RA4/TOCKI.

TOSE : sélectionne le front de la source d'horloge du compteur.

Si TOSE = 1, l'incrémentation se fait quand le signal passe de 0 à 1 (front montant) sur la connexion RA4/TOCKI, et inversement.

PSA : assignement du prédiviseur.

Le prédiviseur effectue une prédivision au niveau du compteur du chien de garde si PSA = 1, et effectue une prédivision pour le TMRO (timer0) si PSA = 0.

PS2,PS1,PS0 : réglage de la prédivision.

Ces trois bits déterminent la valeur de prédivision pour le registre déterminé par le bit PSA. La prédivision est différente pour le compteur (timer0) et pour le chien de garde.

PSA	PS2	PS1	PS0	taux	Durée TMR0
0	0	0	0	2	512 us
0	0	0	1	4	1 024 us
0	0	1	0	8	2 048 us
0	0	1	1	16	4 096 us
0	1	0	0	32	8 192 us
0	1	0	1	64	16 384 us
0	1	1	0	128	32 768 us
0	1	1	1	256	65 536 us

REMARQUE.– Les durées de ce tableau sont valables pour un quartz de 4 MHz.

Chapitre 6

Le reset et les interruptions

6.1. Définition du reset

Après avoir placé le programme en mémoire on peut se demander comment faire pour le lancer et ainsi mettre en route l'application.

L'opération *reset* permet ce démarrage.

Cette procédure d'initialisation interne est déclenchée soit à la mise sous tension du composant c'est le *Power On Reset* (P.O.R.), soit par un niveau bas appliqué sur la connexion MCLR (*Master CLear Reset*).

Celle-ci va charger l'adresse 0x00 dans le compteur de programme. C'est l'adresse de départ de tout programme de PIC. Une fois le compteur de programme chargé, le programme peut alors démarrer.

Tout programme commence à l'adresse 0x00 ; il faut donc écrire en début de programme :

```
org 0x00
goto start
```

REMARQUE.– *start* étant l'étiquette de la première ligne du programme principal.

6.2. Initialisations au moment du Reset

STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C
03h	R/W -0	R/W -0	R/W -0	R -1	R -1	R/W -x	R/W -x	R/W -x

R : lecture, W : écriture. Etat du bit : u : inchangé, 1 : mis à 1 au reset, 0 : mis à 0 au reset, x : inconnu.

Le *reset* initialise un certain nombre de registres afin que rien de dangereux ne puisse arriver aux interfaces avant que le programme de l'utilisateur ne vienne les configurer. Les bits du registre d'état sont positionnés comme l'indique le tableau ci-dessus.

6.3. Les sources de *reset*

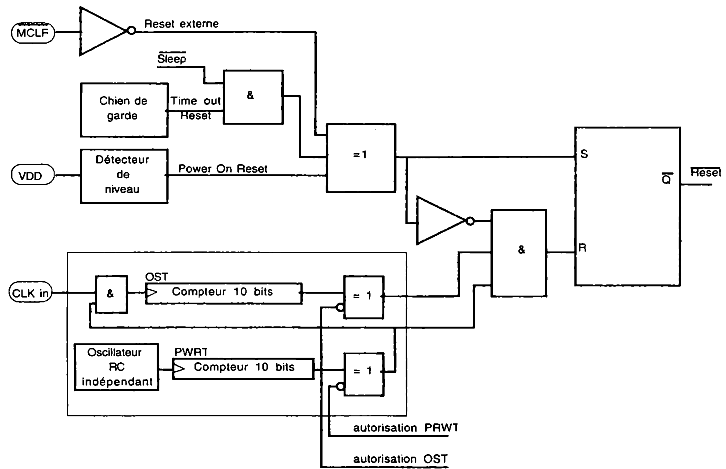


Figure 6.1. *Les sources de* reset

6.3.1. *POR* (Power On Reset) *: la mise sous tension*

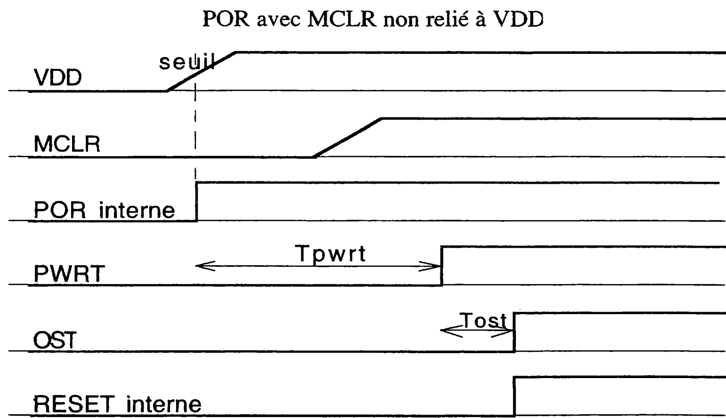


Figure 6.2. Master clear *à l'initialisation interne*

Ce type de *reset* est destiné à initialiser le microcontrôleur.

Le POR est généré grâce à un détecteur de niveau quand la montée de la tension d'alimentation atteint un seuil entre 1,2 V et 1,7 volt.

Ce *reset* déclenche un *timer* indépendant de la vitesse du microcontrôleur (oscillateur RC interne) qui maintient l'état de *reset* (arrêt du microcontrôleur) pendant 72 ms. Ce temps appelé PWRT (*PoWeR-up Timer*) permet à la tension d'alimentation de monter jusqu'à VCC. Ce temps peut être supprimé par le bit PWRTE du registre CONFIG. Ensuite un temps appelé OST (*Oscillateur Start-up Timer*) de 1 024 cycles d'horloge permet à l'horloge à quartz ou à résonnateur de se stabiliser. Cette durée n'est en service que pour le POR et les réveils, et n'est pas utilisé avec l'horloge RC ni avec les autres resets.

Le premier chronogramme montre (voir figure 6.2) que si MCLR (non relié à VDD) est déclenché avant la fin des temps TPWRT + TOST, le reset interne a lieu à la fin de TOST, mais si MCLR arrive après TOST, le microcontrôleur reste à l'arrêt jusqu'au *Master Clear* (voir figure 6.3).

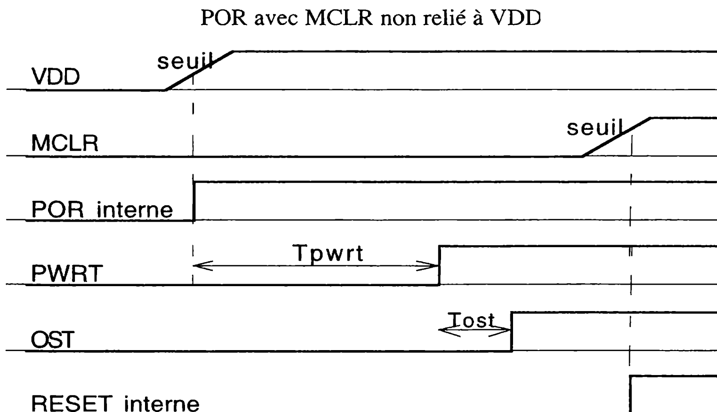
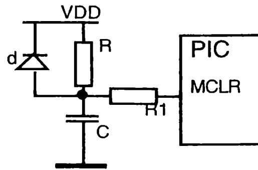


Figure 6.3. *Master clear après initialisation*

Pour profiter du POR il faut placer une résistance entre la connexion MCLR et VDD (figure 6.4). Si après le temps OST, la tension sur MCLR est à 1, le microcontrôleur démarre *(reset* interne), et si MCLR est encore en dessous du seuil, le microcontrôleur attend que MCLR soit arrivé au seuil pour démarrer.

C'est pourquoi le temps de montée de la tension ne doit pas être trop long car sa montée pourrait dépasser le temps PWRT + OST, le microcontrôleur serait mis en marche avant que la tension d'alimentation ne soit établie. Il faut alors placer un circuit RC avec une diode (voir figure 6.4) pour ralentir la montée de la tension sur MCLR.



R<40KΩ, 100<R1<1KΩ
C à déterminer

6.4. *MCLR*

Si la montée de la tension d'alimentation est rapide le *reset* se déclenche après Tost, dans ce cas il n'y a pas de problème (figure 6.5).

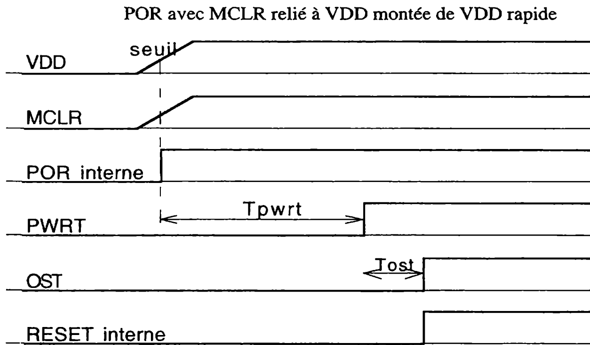


Figure 6.5. Master clear *simultané avec POR*

Si la monte de la tension d'alimentation VDD est très lente, il faut que cette tension dépasse le seuil avant la fin de Tost (figure 6.6).

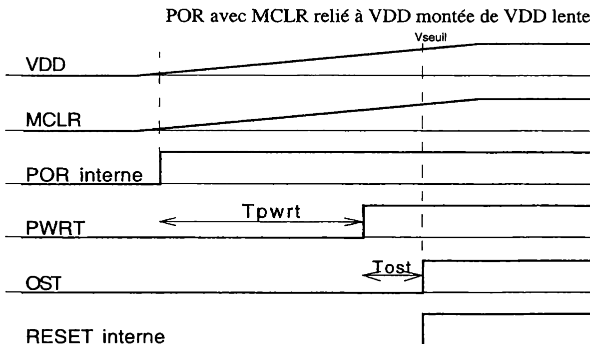


Figure 6.6. *Le reset ne doit se faire que si Vseuil est supérieure ou égale à VDD minimum*

Le compteur de programme est chargé avec l'adresse de départ du logiciel 00h.

Le registre d'état est chargé avec 00011xxx, TO et PD sont à 1 et les bits d'adressage RP1, RP0 et IRP sont à 0. La lecture des bits TO et PD du registre d'état permet de déterminer l'origine du *reset*.

Si la tension d'alimentation peut descendre au-dessous de sa valeur minimum sans atteindre 0, l'utilisateur doit ajouter, par précaution, un dispositif pour que le *reset* ait lieu.

6.3.2. *MCLR reset* (Master CLeaR) *pendant une opération normale*

La connexion MCLR qui permet le déclenchement d'un reset, possède un filtre qui permet d'éliminer les impulsions parasites. La durée minimum de l'impulsion parasite est définie par le constructeur. Dans le cas d'un MCLR, les bits PD et TO ne sont pas modifiés.

Le compteur de programme est alors chargé avec 00h et le registre d'état avec B'000uuuuu'.

6.3.3. *MCLR reset pendant une période de sommeil*

Si un Master Clear intervient pendant que le microcontrôleur est en sommeil, le PC est chargé avec 00h et le registre d'état avec B'00010uuu' (TO=1et PD=0).

6.3.4. *RESET du chien de garde pendant une opération normale*

Le chien de garde est un compteur interne qui fonctionne en permanence. Quand le logiciel fonctionne normalement, il vide ce compteur avec un délai programmé bien précis l'empêchant ainsi de passer de FFh à 0. Si le logiciel est perturbé, le déchargement n'a plus lieu en temps voulu et un niveau 0 est placé pendant 4 cycles d'horloge sur la connexion RESET.

Le PC est chargé avec 00h et le registre d'état avec B'00001uuu' (TO=0 et PD=1).

6.3.5. *RESET du chien de garde pendant une période de sommeil*

Le compteur de programme est incrémenté de 1 et le registre d'état est chargé avec B'uuu00uuu' (TO=0 et PD=0). Le lecteur se reportera au chapitre sur le chien de garde.

6.3.6. *Sortie du sommeil par interruption*

Le PC est incrémenté de 1 (voir le chapitre sur la mise en sommeil du microcontrôleur), si le masque d'interruption général GIE est à 0, le PC est chargé avec PC + 1, et si ce masque est à 1 le PC est chargé avec le vecteur d'interruption 04h.

Adresses	Nom	POWER ON RESET	**MCLR Reset** pendant une opération normale ou pendant le sommeil **Reset par le chien de garde** pendant une opération normale	**Sortie de someil par :** — le chien de garde — une interruption
00h	INDF	---- ----	---- ----	---- ----
01h	TMR0	xxxx xxxx	uuuu uuuu	uuuu uuuu
02h	PCL	0000 0000	0000 0000	PC + 1 [2]
03h	STATUS	0001 1xxx	000q quuu	uuuq quuu
04h	FSR	xxxx xxxx	uuuu uuuu	uuuu uuuu
05h	PORTA	---x xxxx	---u uuuu	---u uuuu
06h	PORTB	xxxx xxxx	uuuu uuuu	uuuu uuuu
08h	EEDATA	xxxx xxxx	uuuu uuuu	uuuu uuuu
09h	EEADR	xxxx xxxx	uuuu uuuu	uuuu uuuu
0Ah	PCLATH	---0 0000	---0 0000	---u uuuu
0Bh	INTCON	0000 000x	0000 000u	uuuu uuuu
80h	INDF	---- ----	---- ----	---- ----
81h	OPTION_REG	111 1111	1111 1111	uuuu uuuu
82h	PCL	0000 0000	0000 0000	PC + 1
83h	STATUS	0001 1xxx	000q quuu	uuuq quuu
84h	FSR	xxxx xxxx	uuuu uuuu	uuuu uuuu
85h	TRISA	---1 1111	---1 1111	---u uuuu
86h	TRISB	1111 1111	1111 1111	uuuu uuuu
87h		---- ----	---- ----	---- ----
88h	EECON1	---0 x000	---0 q000	---0 uuuu
89h	EECON2	---- ----	---- ----	---- ----
0Ah	PCLATH	---0 0000	---0 0000	---u uuuu
0Bh	INTCON	0000 000x	0000 000u	uuuu uuuu
Reg d'état	W	xxxx xxxx	0000 000u	uuuu uuuu

x = inconnu, u = inchangé, - = non utilisé (lu donne 0), q = la valeur dépend des conditions.— 2 : *quand interruption du chien de garde et GIE=1 le PC est chargé avec le vecteur d'interruption*

6.4. Définition d'une interruption

Une interruption comme son nom l'indique, interrompt un programme en cours (appelé aussi programme principal) pour faire exécuter à l'unité centrale (UC) un autre travail (suite d'instructions appelée aussi sous-programme). Celui-ci se termine par une instruction de retour d'interruption (RETFIE) qui permet à l'UC de reprendre le programme principal où il a été quitté.

6.4.1. *Origines d'une interruption*

Sur le boîtier du microcontrôleur la connexion RB0/INT, active au niveau logique 0, a la possibilité de déclencher des interruptions.

Les connexions RB4 à RB7 du PORTB peuvent, quand elles changent d'état demander des interruptions.

Le compteur TMR0 et la fin d'une écriture de données dans la mémoire EEPROM peuvent aussi déclencher des interruptions.

6.4.2. *Opérations effectuées lors d'une interruption*

La demande étant généralement provoquée par un événement extérieur, elle n'est pas synchrone avec le déroulement de l'instruction en cours par conséquent :

– l'instruction en cours est d'abord complètement terminée.

Puis :

– le registre PC contenant l'adresse de retour est seul sauvegardé dans la pile, ce qui impose au programmeur de sauvegarder si cela est nécessaire les valeurs contenues dans l'accumulateur et le registre d'état ;

– le pointeur de pile est décrémenté ;

– le bit GIE (masque général d'interruption) du registre INTCON est mis à 0, il interdit ainsi la prise en compte d'une nouvelle demande ;

– l'adresse de départ du sous-programme d'interruption 0x04 est chargée dans le PC puis le sous-programme d'interruption est effectué.

Un temps d'attente de 3 ou 4 cycles d'instruction après l'apparition de la demande permet toutes ces opérations et notamment le chargement du PC.

Quand le sous-programme d'interruption est terminé :

– celui-ci se termine obligatoirement par l'instruction RETFIE.

Cette instruction permet :

– à l'unité centrale de récupérer l'adresse de retour c'est à dire au compteur de programme de se charger avec l'adresse de l'instruction du programme principal qu'il devait exécuter au moment de la demande d'interruption ;

– au pointeur de pile d'être incrémenté ;
– la remise à 1 de GIE.

6.4.3. *Les interruptions : généralités*

Une interruption ne peut interrompre un sous-programme d'interruption.

Les interruptions sont masquables par le bit GIE (*General Interrupt Enable*) du registre INTCON. Ce masque à 1 autorise et à 0 l'interdit toutes les interruptions. GIE est mis à 0 par le *reset.*

La ligne INT peut être rendue sensible à un front descendant en plaçant à 0 le bit INTEDG (INTeruption EDGe) du registre OPTION et inversement.

6.4.4. *Détermination de l'origine de la demande d'interruption*

Il y a plusieurs origines possibles et c'est au programmeur de trouver l'origine de la demande. Pour cela il faut lire les indicateurs (flags) qui indiquent chacun une source de la demande. Ces bits, situés dans le registre INTCON, passent à 1 quand une demande est apparue.

INTCON	GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
0Bh et 8Bh	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -x

R : lecture, W : écriture, x : inconnu, u : inchangé, 1 : mis à 1 au reset, 0 : mis à 0 au reset

Ces bits sont :

TOIF (*Timer Overflow Interrupt Flag*) : ce bit est mis à 1 quand le compteur TMR0 déborde c'est-à-dire quand son contenu passe de FFh à 00h ;

INTF (*INT Flag*) : ce bit est mis à 1 quand un front est détecté sur l'entrée RB0/INT même si les interruptions ne sont pas autorisées ;

RBIF (*port change Interrupt Flag*) : ce bit est mis à 1 quand un changement d'état est détecté sur une des connexions RB4 à RB7 du port B. Ces quatres connexions sont indissociables, c'est-à-dire que l'événement est pris en compte quelle que soit la connexion stimulée.

Tous les indicateurs activés doivent être remis à 0 durant le sous-programme d'interruption.

6.4.5. *Les masques d'interruption*

Pour qu'une interruption puisse être prise en compte il faut que le GIE (masque général) autorise les interruptions en étant à 1 et que le masque correspondant à cette interruption soit à 1. Les masques d'interruptions sont situés dans le registre INTCON.

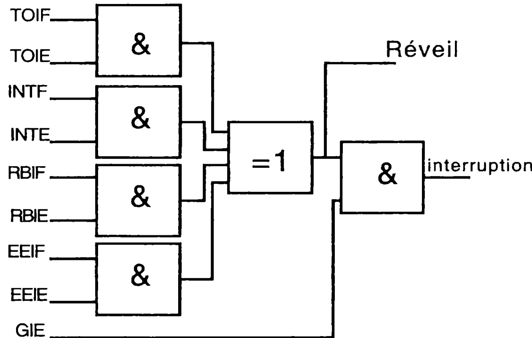


Figure 6.7. *Schéma logique des interruptions*

Ce registre se situe à l'adresse 0Bh ou encore à l'adresse 8Bh. Les masques sont positionnables par le programmeur, ils sont mis à 0 dès le début de l'interruption et sont remis à 1 par l'instruction de retour d'interruption RETFIE.

GIE : (Global Interrupt Enable) interdit toutes les interruptions tant qu'il est à 0.

Ce bit est mis à 0 automatiquement pendant un sous programme par l'unité centrale pour éviter de perturber le déroulement de celui-ci. Il est remis à 1 par l'instruction RETFIE.

EEIE (*EEprom write complete Interrupt Enable*) : autorise quand il est à 1 la demande d'interruption quand l'opération d'écriture dans l'EEPROM est terminée.

TOIE (*Timer Overflow Interrupt Enable*) : masque qui autorise à 1 le déclenchement d'une interruption quand il y a débordement du compteur TMR0.

INTE (*INT Enable*) : masque associé à la connexion RB0/INT de demande externe d'interruption. Autorise l'interruption quand il est à 1.

RBIE (*Port Change Interrupt*) : autorise quand il est à 1 les interruptions demandées par un changement d'état de l'une des connexions RB4 à RB7 du port B.

6.5. Résumé pour la mise en œuvre

L'utilisateur doit :

– démasquer les interruptions prévisibles dans le programme principal ;

– faire commencer son programme d'interruption à l'adresse 0x04 ;

– dans le programme d'interruption il doit :

- chercher quelle est la source de la demande ;

- sauvegarder certaines données comme le contenu de l'accumulateur W et le contenu du registre d'état ;

- remettre à 0 le flag correspondant afin d'éviter un redéclenchement d'interruption à chaque retour ;

– ne pas faire plus de 7 imbrications de sous-programmes car la pile ne possède que 8 emplacements.

6.6. Exemple de mise en œuvre d'une interruption externe

Un bouton poussoir est câblé par l'intermédiaire d'une bascule antirebond sur l'entrée RB0/INT d'un microcontrôleur et une led en série avec une résistance est branchée sur la connexion PA3. A chaque appui sur le poussoir, une interruption est déclenchée et le programme d'interruption provoque un changement d'état de la ligne PA3 (led allumée ou éteinte).

```
Source de ce programme :
     LIST p=16F84
     #include "P16F84.INC"

OPTIONVAL EQU B'01000000'     ; registre OPTION determine le front
                              ; de pulse déclenchement de la demande
                              ; d'interruption.
INTCONVAL EQU B'10010000'     ; registre INTCON permet d'autoriser
                              ; les interruptions sur RB0 grâce au
                              ;masque INTE
#DEFINE LED PORTA,3           ;

     CBLOCK 0x00C             ; definition d'une zone de variable
     W_temp : 1               ; on reserve un emplacement memoire
                              ; pour sauvegarder W pendant le
                              ; programme d'interruption
     STATUS_temp : 1          ; reserve pour sauvegarder le registre
                              ; d'état
     ENDC                     ; fin de zone des variables

     ORG 0x00
     GOTO START
; Sous programme d'interruption
```

```
    ORG 0x04
;sauvegarde des registres
    movwf W_temp        ; sauve W dans W_temp
    swapf STATUS,w      ; echange
    movwf STATUS_temp   ; sauve dans STATUS_temp
    call INTBR0         ;  programme actif de l'interruption
    swapf STATUS_temp,W
    movwf STATUS        ; recuperation de STATUS
    swapf W_temp,f
    swapf W_temp,W      ; recuperation de W
    retfie              ; retour d'interruption

; Programme principal
; il n'a aucune action directe sur le fonctionnement mais il
; initialise les registres


START bcf STATUS,RP0    ; bloc mémoire 0
    bsf LED             ; met la ligne RA3 a 1
    bcf STATUS,RP0      ; bloc mémoire 0
    clrf TRISA          ; met le port A en sortie
    movlw INTCONVAL     ; initialise le registre INTCON
    movwf INTCON
    bsf STATUS,RP0      ; bloc 1
    movlw OPTIONVAL
    movwf OPTION_REG    ; initialise le registre OPTION
    bcf STATUS,RP0
BOU NOP                 ; non operation
    NOP                 ; boucle d'attente
    NOP
    GOTO BOU            ; infinie



INTBR0    bcf STATUS,RP0 ; passe au bloc 0
    movlw B'00001000'
    xorwf LED           ; inversion de la ligne avec un OUexclusif
    bcf INTCON,INTF     ; efface le flag correspondant a RB0
    return
    END
```

Chapitre 7
Les interfaces parallèles

Le microcontrôleur doit pouvoir effectuer des échanges avec l'extérieur, donc transmettre des informations sous forme parallèle (tous les bits sont transmis simultanément). Pour cela il dispose de ports qui peuvent émettre et recevoir (bidirectionnels).

Ces interfaces ou ports sont constitués de registres, chacun se comportant comme une case mémoire et par conséquent dispose d'une adresse. Chaque registre a un rôle précis ; programmer le fonctionnement de l'interface, contenir des données reçues ou à émettre ou encore déterminer le sens de transfert des données.

Après le *reset* (MCLR) toutes ces connexions sont en entrées.

7.1. Le port A

Le port A à l'adresse 0x05 dispose de 5 lignes bidirectionnelles disposant chacune d'un « latch », compatibles TTL en entrée et CMOS en sortie. En sorties elles peuvent débiter jusqu'à 20 mA.

Le port A est associé bit à bit à un registre de direction TRISA (*TRansfert Input Set*), situé à l'adresse 0x85 donc dans le bloc mémoire 1.

Si un bit de TRISA est à 1, la ligne correspondante du port est en entrée, ce qui sous-entend que la ligne du port est en état de haute impédance (voir figure 7.1).

Si un bit de TRISA est à 0, la ligne de même numéro est en sortie et met le contenu de la bascule mémoire (*latch*) correspondante sur la ligne de sortie. On peut donc placer une donnée en mémoire et la faire apparaître en sortie par la commande du registre de direction.

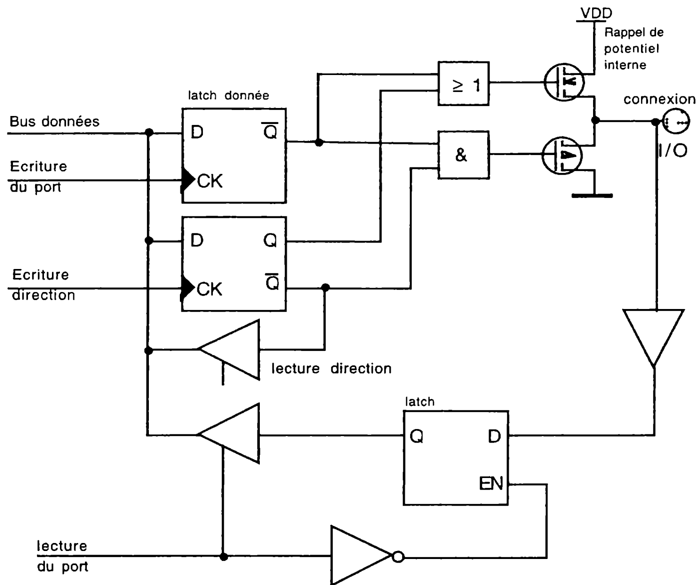


Figure 7.1. *Structure des connexions RA0 à RA3*

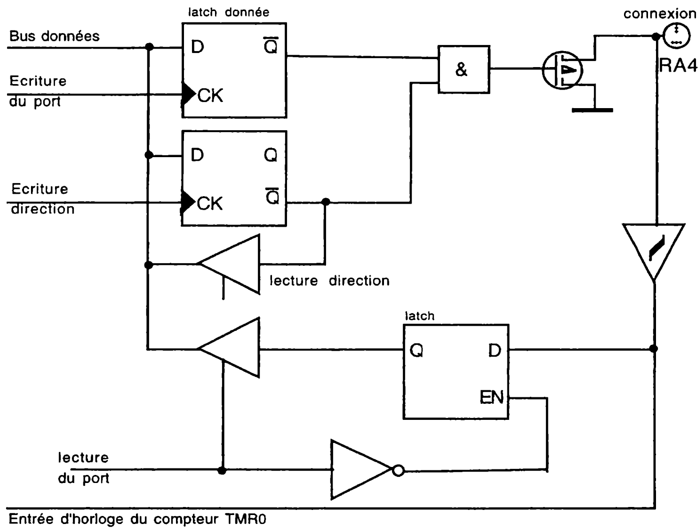


Figure 7.2. *Structure de la connexion RA4*

Au reset du microcontrôleur, toutes les connexions sont en entrées pour éviter d'imposer des niveaux sur les composants qui y sont cablées.

Toute écriture dans le port correspond à :
- une lecture de tout le port ;
- une modification de la valeur ;
- une écriture de cette nouvelle valeur.

La ligne RA4 est multiplexée, dotée d'une bascule de Schmitt, elle permet aussi le comptage d'événements extérieurs avec le compteur TMR0 (voir chapitre suivant). La bascule de Schmitt permet à cette entrée d'être indépendante du temps de montée des signaux.

RA4 est aussi une sortie à drain ouvert.

Exemple d'utilisation du port A

Faire passer la ligne RA0 à 1 et la ligne RA1 à 0.

```
CLRF PORTA          ; initialise le latch du port A
BSF STATUS,RPO      ; passe au bloc 1 pour accèder à TRISA
MOVWL 0xFC          ; met FCh dans W
MOWWF TRISA,f       ; met FCh dans TRISA pour mettre les
                    ; connexions RA0 et RA1 en sorties; RA0 et
                    ; RA1 sont à 0 et RA2 à RA4 sont
                    ; en entrées donc en haute impédance.
BCF STATUS,RP0      ; passe au bloc mémoire 0
BCF PORTA,1         ; met RA1 à 0
BSF PORTA,0         ; met RA0 à 1
```

7.2. PORT B

Le port B, à l'adresse 0x06, est constitué de 8 entrées/sorties, dont le sens est déterminé par le registre TRISB situé à l'adresse 0x86.

Les sorties sont compatibles TTL et les entrées en Cmos sont compatibles TTL.

Chaque bit de TRISB quand il est à 1 met la ligne correspondante en état de haute impédance, autrement dit cette ligne peut être utilisée en entrée, et quand il est à 0, fait passer le contenu du *latch* sur la connexion ; la ligne est alors en sortie.

Chaque ligne du port peut se voir attribuer une résistance de rappel de potentiel interne par la programmation du bit RBPU du registre OPTION (81h). La mise en sortie du port et le RESET suppriment automatiquement ces résistances.

7.2.1. *Précautions d'utilisation du port B*

Toute opération d'écriture est en fait constituée d'une lecture entière du port, d'une modification de donnée suivie d'une écriture.

7.2.1.1. *Problème de la bascule mémoire*

La présence de la bascule mémoire (*latch*) située sur chaque connexion peut, si on n'y prend garde, donner des résultats inattendus donc préjudiciables.

La modification d'un bit provoque d'abord la lecture du port donc de toutes les lignes y compris celles qui sont en entrées. Le processeur lit donc l'état de la ligne à un moment donné, puis quand il va écrire sur le port il va recopier cet état (contenu du *latch* dans la bascule correspondante. Cela ne pose aucun problème tant que la ligne considérée est en entrée, mais quand celle-ci va passer en sortie elle va mettre le contenu de la bascule sur la ligne ce qui n'est pas forcément souhaité et peut poser problème vis à vis du composant externe (voir figure 7.3.).

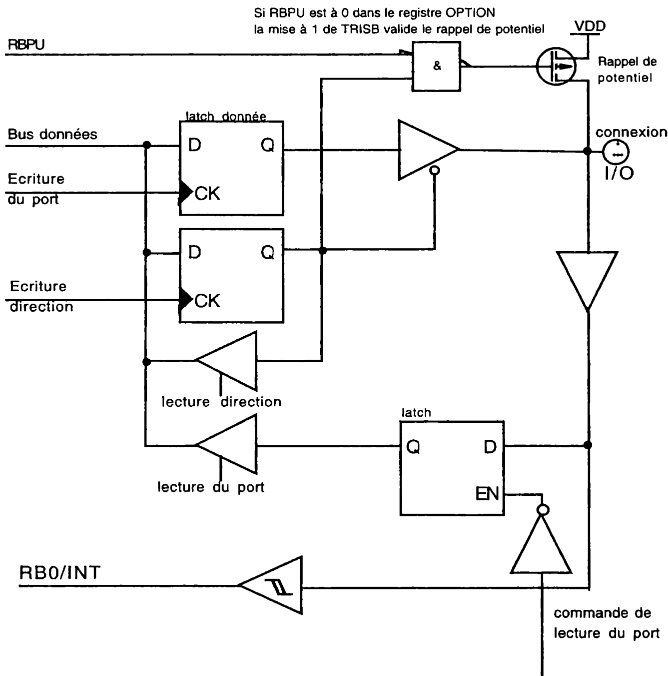


Figure 7.3. *Connexions RB0 à RB3 du port B*

EXEMPLE.– Les lignes 0 à 3 du portB sont placées en entrées, les lignes 4 à 7 sont en sorties et les lignes RB6 et RB7 sont reliées à VCC par des résistances de rappel de potentiel externes.

```
                     bascules    connexions
                     internes    externes
BCF PORTB,7          01-- ----   11-- ---- ; mise à 0 du bit 7
BCF PORTB,6          10-- ----   11-- ---- ; mise à 0 du bit 6
BSF STATUS,RP0                   ; passe au bloc 1 pour accèder
                                 ; à TRISB (86h)
BCF TRISB,7          10-- ----   11-- ----; connexion 7 en sortie
BCF TRISB,6          10-- ----   10-- ----; connexion 6 en sortie
```

Quand on met la connexion 7 en sortie rien ne change car dans la bascule mémoire il y avait un 1 mémorisé précédemment, et quand on met la ligne 6 en sortie, le 0 qui avait été mémorisé est mis en place sur la ligne.

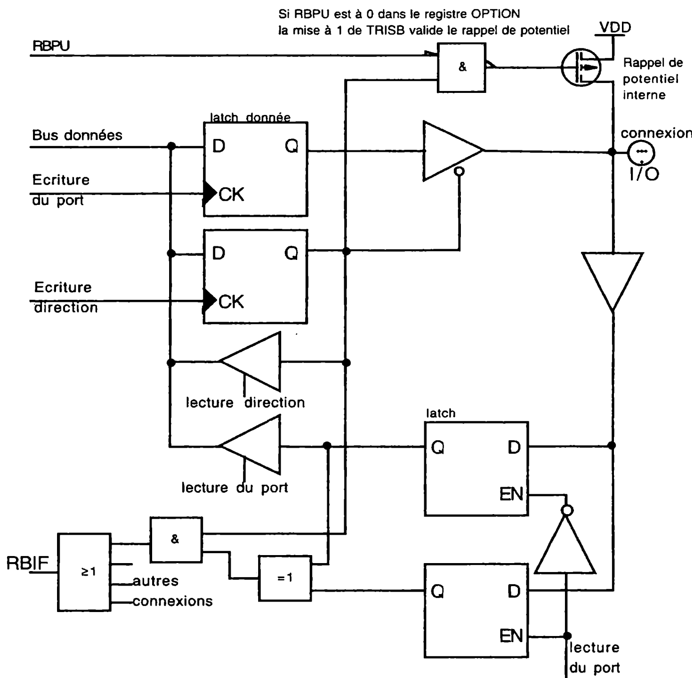


Figure 7.4. *Connexions RB4 à RB7 du port B*

7.2.1.2. *Problème de la structure de sortie*

Une connexion en train de passer d'un état bas à un état haut ne doit pas être activée depuis un composant externe qui fait changer de niveau en même temps, car le courant dans la connexion, quasiment en court circuit, peut endommager le microcontrôleur.

7.2.1.3. *Problème du temps d'établissement des lignes en sortie*

Le microcontrôleur fonctionne à raison d'une instruction par cycle machine, comme une opération sur un port apparaît à la fin d'un cycle, le signal se met en place sur la ligne pendant le début du cycle suivant (instruction suivante). Il est donc nécessaire de laisser le temps au signal de se stabiliser avant de vouloir l'utiliser. Par exemple après une écriture sur un port il est judicieux de placer un NOP avant d'effectuer une relecture de ce port.

7.2.2. *Les interruptions sur le port B*

Les connexions RB4 à RB7 peuvent aussi être utilisées pour demander une interruption quand elles sont programmées en entrées. RB0 est aussi une entrée de demande d'interruption (voir le chapitre concernant les interruptions).

Les demandes d'interruption sont effectuées quand une modification de l'état de l'une de ces lignes en entrées est détectée. La valeur de la ligne est comparée avec l'ancienne valeur qui a été mémorisée dans la bascule d'entrée lors de la précédente lecture (voir figure 7.4).

Les demandes d'interruption qui peuvent être générées par l'une des quatre connexions sont regroupées dans une porte OU qui met à 1 l'indicateur RBIF du registre d'état. C'est ce bit RBIF qui peut alors déclencher la demande d'interruption ou réveiller le microcontrôleur si il est en sommeil (voir le chapitre concernant la mise en sommeil du microcontrôleur).

Le bit RBIF peut être remis à 0 par programme ou par la lecture ou l'écriture du port B qui met fin automatiquement au procédé de détection de changement d'état.

Cette procédure de détection de changement d'état est très bien adaptée pour la gestion d'un clavier. En effet la détection de l'appui sur une touche peut permettre le réveil du microcontrôleur et ensuite la lecture de la valeur de la touche.

Le lecteur trouvera une application de cette interruption pour la gestion d'un clavier dans le chapitre 10 sur la mise en sommeil du microcontrôleur.

7.3. Application

On veut afficher sur le port A la valeur BCD d'un nombre obtenu grâce à une roue codeuse hexadécimale branchée sur les bits de poids faible du port B.

Cet exercice permet de découvrir :

- l'usage des ports A et B ;
- l'usage du bit DC du registre d'état ;
- un mode de conversion d'un nombre binaire en BCD (par addition de 6).

Source du programme :

```
    list P=16F84
    __CONFIG _CP_OFF & _WDT_OFF & _XT_OSC
    include <p16f84.inc>

    CBLOCK 0x0c
    sauv : 1            ; registre pour une sauvegarde provisoire
    ENDC

    org 0x00
    goto INIT

;initialisation et programme principal

INIT bcf STATUS,RP0     ; passe au bloc 0
    movlw 0x00
    movwf PORTB         ; charge latch port B avec 00
    movwf PORTA         ; charge latch port A avec 00
    bsf STATUS,RP0      ; passe au bloc 1
    movlw 0xff          ; en entrée =>niveau 1
    movwf TRISB         ; port B en entrées
    movlw 0x00          ; port A en sorties
    movwf TRISA

;Procedure lecture

LECTURE bcf STATUS,RP0  ; bloc 0
    movf PORTB,W        ; lecture du quartet de poids faible
    andlw 0x0f          ; récupère les bits de poids faible
    movwf sauv          ; sauvegarde
    addlw 6             ; ajoute 6 à w
    btfss STATUS,DC     ; DC est-il à 1?
    movf sauv,W         ; si non récupère la lecture directe
    movwf PORTA         ; affichage sur PORTA
    goto LECTURE        ; recommence

    END
```

Chapitre 8
La mesure du temps

8.1. Le compteur temporisateur

Un temporisateur dans un système informatique est destiné à créer une base de temps ce qui permet de libérer le programme de cette tâche.

Dans le PIC, le temporisateur est un registre de largeur 8 bits qui compte des impulsions d'horloge et qui fait passer un indicateur TOIF à 1 quand il passe de FFh à 0 (débordement).

L'horloge qui l'incrémente peut être d'origine interne ou externe, et le débordement peut provoquer une demande d'interruption. Sa fréquence peut être prédivisée par un nombre allant jusqu'à 256.

L'utilisation de l'horloge interne est évidemment destinée à la mesure du temps mais on peut aussi grâce à l'entrée d'horloge externe utiliser le registre pour compter des événements.

8.2. Choix de l'horloge

Le mode de fonctionnement compteur ou temporisateur est déterminé par l'origine de l'horloge. C'est le bit TOCS (*Timer 0 Clock Source bit*) du registre OPTION (81h) qui permet de faire ce choix.

Si TOCS est à 0, l'horloge qui incrémente le compteur est celle du quartz divisée par quatre, donc c'est l'horloge du microcontrôleur.

Si TOCS est à 1, l'incrémentation se fait par les fronts montants ou descendants des signaux reçus sur la connexion RA4/TOCKI. Le front actif de ces signaux est déterminé par le bit TOSE (*Timer 0 Source Edge bit*) du registre OPTION. Si TOSE est à 0, ce sont les fronts montants qui sont actifs et inversement.

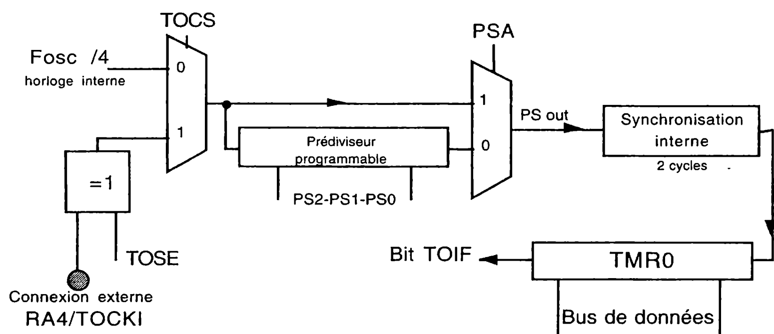


Figure 8.1. *Schéma de la fonction compteur*

Si le bit PSA du registre OPTION est à 0 ce sont les signaux issus du prédiviseur (PS out) qui sont synchronisés avec l'horloge interne du microcontrôleur pour incrémenter le compteur TMR0. Quand celui-ci déborde le bit TOIF passe à 1. Le registre compteur TMR0 est accessible par le bus de données. Pour compter des évènements extérieurs par TOCKI ; TOCS à 1 et PSA à 1.

8.3. Le diviseur programmable

Le diviseur est un compteur asynchrone qui est commun au temporisateur et au chien de garde mais ne peut être utilisé par les deux simultanément. Ce compteur n'est pas accessible au programmeur ; il ne peut être ni lu ni écrit.

Sa mise en service dépend du bit PSA du registre OPTION (81h) qui permet de l'attribuer soit au temporisateur soit au chien de garde.

OPTION_REG	RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0
81h	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1

Si PSA est à 0, le prédiviseur affecté au temporisateur, peut diviser la fréquence d'horloge de 1/2 à 1/256, ce taux de division est programmable grâce aux trois bits PS0, PS1 et PS2 du registre OPTION. Si PSA est à un, le diviseur est affecté au chien de garde et peut diviser avec un taux de 1/1 à 1/128.

Quand il est assigné au compteur, toutes les instructions relatives au TMR0 remettent à 0 le prédiviseur. Quand il est affecté au chien de garde, l'instruction CLRWDT (*clear watchdog*) remet à 0 le diviseur.

Tableau de programmation pour un quartz de 4 MHz :

PSA	PS2	PS1	PS0	taux	Durée TMR0
0	0	0	0	2	512 us
0	0	0	1	4	1 024 us
0	0	1	0	8	2 048 us
0	0	1	1	16	4 096 us
0	1	0	0	32	8 192 us
0	1	0	1	64	16 384 us
0	1	1	0	128	32 768 us
0	1	1	1	256	65 536 us

Avec un quartz de fréquence 4 MHz, on peut obtenir, pour une incrémentation complète du temporisateur, une durée allant de 512 us à 65 536 us.

8.4. Les interruptions

Pour détecter le débordement (passage de FFh à 0) du registre TMR0, il suffit de scruter le passage à 1 du bit TOIF (*Timer Overflow I Flag*) dans le registre INTCON.

INTCON	GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
0Bh	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -x

R : lecture, W : écriture, x : inconnu, u : inchangé, 1 : mis à 1 au reset, 0 : mis à 0 au reset

Pour réaliser des bases de temps il est très intéressant de travailler avec les interruptions. Quand le registre TMR0 déborde il fait passer le bit TOIF à 1, ce qui permet d'appeler un sous-programme d'interruption si le bit TOIE (*Timer Overflow Interrupt Enable*) du registre INTCON (0Bh) est à 1. On peut masquer les interruptions provoquées par TMR0 en mettant à 0 ce masque TOIE.

REMARQUE.– Il faut absolument remettre TOIF à 0 dans le sous-programme d'interruption, sinon au moment du retour d'interruption RETFIE, le bit TOIE étant remis automatiquement à 1, une interruption serait redéclenchée et ceci de façon infinie.

8.5. Exemples d'utilisation du compteur

8.5.1. *Réalisation d'un compteur de pièces programmable*

Un capteur est connecté par l'intermédiaire d'une bascule RS antirebond à la ligne PA4/TOCKI ; il envoie un signal front descendant à chaque passage d'une pièce. Le compteur actionne un vérin une fois que le nombre de pièces désiré est passé devant le capteur, ce vérin est équipé d'un capteur fin de course qui permettra le redémarrage d'un nouveau cycle de comptage.

Le capteur fin de course est branché sur la ligne RA1 et délivre un niveau 1 quand il est activé et le vérin est commandé par la ligne RA0.

Solution : activer TOCS pour mettre en service le compteur et TOSE pour définir le front actif sur l'entrée. Pour compter le nombre de pièces on retranchera ce nombre à 256, ainsi quand il sera atteint, le contenu du TMR0 passera à 256 donc passera de FFh à 0.

Etude des registres :

– le registre OPTION va permettre de définir le mode de fonctionnement donc tous les bits sont mis à 0 sauf les bits TOCS, TOSE et PSA ;

– dans le registre INTCON qui gère les interruptions, l'indicateur TOIE sera chargé avec 0.

Source du programme :

```
    list p=16F84
    include "p16F84.inc"

#define VERIN PORTA,0
#define FINCOURS PORTA,1

OPTIONVAL equ B'00111000'      ; réglages
INTMASK equu B'00000000'       ; interruptions
NBPIECE equ 2

    org 0x00
    goto START

START bsf STATUS,RP0       ; passe en banque 1
    movlw OPTIONVAL
    movwf OPTION_REG       ; mise en place des reglages
    movlw INTMASK
    movwf INTCON           ; masque interruptions
    movlw B'00010010'
    movwf TRISA            ; initialise port A
BOU2
    bcf STATUS,RP0         ; passe en banque 0
    movlw 256-NBPIECE      ; mise en place
    movwf TMR0             ; du nombre de pieces
    bcf INTCON,T0IF        ; RAZ du flag
    bsf VERIN              ; verin en position initiale
BOU btfss INTCON,T0IF      ; test TMR0 a deborde ?
    goto BOU               ; si non recommence
    bcf VERIN              ; si oui avance verin
BOU3 btfss FINCOURS        ; verin en fin de course ?
```

```
    goto BOU3          ; si non attendre
    goto BOU2          ; si oui nouveau cycle
    END
```

8.5.2. *Réalisation d'une attente avec TMR0 et interruption*

On veut faire clignoter une led avec une période d'environ une seconde.

La led est branchée sur la ligne RA2. Le quartz du microcontrôleur a une fréquence de 4 MHz.

Solution

Le timer déborde au bout de 256 cycles d'horloge si on n'emploie pas le prédiviseur et au bout de 65 536 cycles avec le prédiviseur employé au maximum. Un cycle est égal à 1 us dons il faudra recommencer l'opération 8 fois pour obtenir une durée d'environ 0,5 s.

Pour cela il faut configurer le registre OPTION, le registre INTCON et définir une variable NBPAS dans un registre pour compter les 8 passages.

Le programme principal initialise les registres puis est bouclé sur lui-même pour ne rien faire. A chaque fois que le registre TMR0 déborde, il déclenche une interruption qui décrémente le compteur de passage. Si le compteur atteint 0, l'état de la ligne de la led est inversé, et le compteur est rechargé pour la prochaine temporisation, sinon on retourne au programme principal en attendant la prochaine interruption.

Etude des registres

INTCON :

GEI = 1, EEIE=0, TOIE=1, INTE=0, RBIE=0, TOIF=0, INTF=0, RBIF=0 pour autoriser les interruptions.

OPTION

RBPU=0, INTEDG=0, TOCS=0, TOSE=0, PSA=0, PS2=1, PS1=1, PS0=1 pour prédiviser par 256 l'horloge du microcontrôleur.

Source du programme

```
;Fait clignoter une led sur RA2
;Utilise le TMR0 prédivisé par 128, huit fois entre chaque inver-
sion.

    list p=16F84
    #include <p16F84.inc>
    __CONFIG   _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC
```

```
INTMASK equ B'10100000'       ; réglages interruptions GIE, TOIF
OPTION_VAL equ B'10001111'    ; réglages tmro /128, pullup off
NBPAS equ 8                   ; définition du nombre de passages

#DEFINE LED PORTA,2

     cblock 0x0C
     NPAS : 1       ; réservation pour compteur de passages
     endc

     org 0x00
     goto INIT

;sous programme d'interruption
     org 0x04

INTER     BCF STATUS,RP0
     decfsz NPAS,f        ; le nombre de passages est-il atteint ?
     goto FIN             ; si non retour d'interruption
     movlw  B'00000100'   ; choix de la ligne à inverser
     xorwf LED            ; le OUexclusif inverse le bit
     movlw NBPAS
     movwf NPAS           ; recharge le compteur de passages
FIN  BSF STATUS,RP0
     bcf INTCON,T0IF      ; flag TOIF remit à 0
     RETFIE

; programme principal
INIT bsf STATUS, RP0      ; passe au bloc 1
     movlw  OPTION_VAL
     movwf OPTION_REG     ; mettre en place les réglages
     movlw INTMASK
     movwf INTCON         ; mettre en place les interruptions
     movlw 0
     movwf TRISA          ; mise en sortie du PORTA
     bcf STATUS,RP0       ; passe au bloc 0
     movlw NBPAS
     movwf NPAS           ; chargement  du compteur de passage
     clrf PORTA


BOU
     goto BOU       ; programme d'attente sans action
     END
```

Chapitre 9
La surveillance de fonctionnement

Le chien de garde est un système de sécurité qui surveille le bon déroulement d'un programme.

9.1. Généralités

Le chien de garde (*watchdog*) est structuré autour d'un compteur permanent dont l'horloge est à base d'un oscillateur RC interne. Cette horloge fonctionne en permanence même quand celle du microcontrôleur est arrêtée en mode sommeil. Hors sommeil, si ce compteur n'est pas remis à zéro avant qu'il ne déborde, il provoque un *reset* du microcontrôleur donc le redémarrage du programme à l'adresse 0x00.

Pour l'utilisation du chien de garde l'utilisateur doit donc prévoir dans son logiciel, cette remise à zéro périodique grâce à l'instruction CLRWDT (*CLeaR WatchDog Timer*).

La durée normale du remplissage de ce compteur est d'environ 18 ms, toutefois cette durée dépend de l'alimentation et de la température, il est préférable alors de prendre comme valeur mini 7 ms (voir tableau de programmation page 105).

9.2. Mise en œuvre

Son utilisation est à prévue pour :

– vérifier si un programme n'a pas été modifié par un parasite ;

– vérifier que le programme n'est pas dans une boucle infinie ou qu'un cycle de fonctionnement n'a pas été prévu en cas d'utilisation sur une machine d'états ;

– réveiller un microcontrôleur en mode sommeil (oscillateur arrêté).

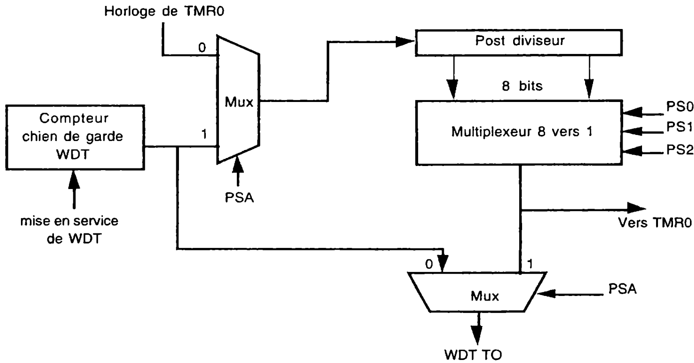


Figure 9.1. *Schéma fonctionnel du chien de garde*

La mise en service du chien de garde se fait par la mise à 1 du bit WDT du registre CONFIG ; il faut donc prévoir :

__CONFIG _WDT_ON dans la programmation du microcontrôleur.

Si PSA est à 1, la sortie du compteur indépendant WDT est branché sur le diviseur, dans ce cas celui-ci est appelé post-diviseur. Les bits PS0 à PS2 déterminent le taux de division. PSA étant à 1, le signal issu du multiplexeur détermine l'état du bit TO (*Time Out*) du chien de garde. On peut remarquer que si PSA est à 0, c'est l'horloge du TMR0 qui arrive sur le diviseur, on l'appelle alors prédiviseur puisque ensuite la sortie du multiplexeur est connectée au compteur TMR0.

OPTION_REG	RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0
81h	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1

La mise en service et la non-remise à zéro du compteur du chien de garde crée des redémarrages permanents (*reset*) et la mise à 0 du bit TO (*Time Out*) du registre d'état.

STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C
03h ou 83h	R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-1x

Les instructions CLRWDT et SLEEP provoquent la remise à 0 du compteur permanent et du post-diviseur, et le bit TO est mis à 1. Elles doivent donc intervenir avant le débordement pour éviter le RESET.

Dans ce cas le programme recommence à l'adresse 0x00. L'utilisateur peut prévoir un test au début de son logiciel pour déterminer si le redémarrage est un POR ou s'il est dû à un défaut détecté par le chien de garde et dans ce cas prévoir une intervention.

9.3. Durée de la protection

Le registre du chien de garde met normalement 18 ms pour se remplir grâce à un oscillateur RC interne, mais on peut ajouter un post-diviseur pour augmenter cette durée jusqu'à plus de deux secondes et ceci indépendamment de la fréquence du quartz du microcontrôleur.

Le post-diviseur est connecté par le bit PSA et le taux de division dépend des bits PS0, PS1 et PS2 du registre OPTION.

Tableau de programmation :

PSA	PS2	PS1	PS0	taux	Durée de WDT
1	0	0	0	1	18 ms (7 ms)
1	0	0	1	2	36 ms (14 ms)
1	0	1	0	4	72 ms (28 ms)
1	0	1	1	8	144 ms (56 ms)
1	1	0	0	16	288 ms (112 ms)
1	1	0	1	32	576 ms (224 ms)
1	1	1	0	64	1,152 s (448 ms)
1	1	1	1	128	2,304 s (996 ms)

9.4. Précautions d'emploi

9.4.1. *Ne pas mettre l'instruction CLRWDT dans un sous-programme d'interruption*

Quand le sous-programme d'interruption est terminé le programme principal reprend à l'endroit où il a été quitté même si celui-ci n'est pas à un endroit « normal ». Si l'instruction CLRWDT est dans le sous-programme elle risque d'être exécutée même si le programme principal est « planté ».

9.4.2. *Pour passer du prédiviseur du TMR0 au post-diviseur du chien de garde*

Pour éviter un reset indésiré il faut respecter les lignes de programme suivantes :

– pour faire passer le diviseur du TMRO au chien de garde :

```
bcf STATUS,RP0        ; passer au bloc 0
clrf TMRO             ; mise à 0 du temporisateur et du pré-
                      ; diviseur
bsf STATUS,RP0        ; passer au bloc 1
clrwdt                ; mise à 0 du chien de garde
movlw b'xxxx1xxx'     ; nouvelle prédivision
movwf OPTION_REG      ; dans le registre option
bcf STATUS,RP0        ; passer au bloc 0
```

– pour faire passer le diviseur du chien de garde au temporisateur :

```
clrwdt                  ; mise à 0 du chien de garde et du
                        ; diviseur
bsf STATUS,RP0          ; passer au bloc 1
movlw b'xxxx0xxx'       ; sélection du TMR0 et du taux de
                        ;division
movwf OPTION_REG
bcf STATUS,RP0          ; passer au bloc 0
```

9.5. Exemple d'application : sécurité sur un moteur

Un capteur placé sur un axe de moteur délivre à chaque rotation un niveau logique 0 pendant 1 ms sur la connexion RB1du port B, la ligne RB0 commande la rotation du moteur quand elle est à 1 et une led branchée sur RB2 allumée par un niveau 0 signale un défaut de rotation du moteur (ralentissement). Le moteur doit être à sa vitesse normale avant la mise en service de la sécurité.

Le moteur tourne à 500 t/mn donc le capteur doit envoyer au microcontrôleur une impulsion toutes les 6,25 ms, en cas d'arrêt ou de ralentissement le microcontrôleur signale le problème en allumant la led et en arrêtant le moteur.

Solution

Le microcontrôleur détecte le niveau fourni par le capteur et remet à 0 le WDT puis après un temps d'attente légèrement inférieur à une période de rotation recommence la détection. Si le moteur ralenti, le temps entre deux détections étant trop long, le chien de garde va déclencher un RESET. Le programme en redémarrant va tester le bit TO du registre d'état. Si celui-ci est à 0 (le compteur du chien de garde a débordé), cela veut dire que le moteur a ralenti, dans ce cas la led est allumée et le moteur est arrêté.

Etude des registres

Il faut mettre les lignes du port B en entrées sauf les bits RB0 et RB2, donc 0xFA dans TRISB.

Il faut mettre RBPU à 0 pour avoir les résistances de rappel de potentiel en service, PSA à 1 et PS0 à PS2 à 0 pour mettre le post-diviseur en service avec une division de 1 (7 ms environ), donc 0x78 dans OPTION_REG.

Source du programme pour un quartz de 4 MHz

```
LIST        p=16F84         ; microcontrôleur
#include "p16F84.inc"       ; fichier de définitions

__CONFIG    _CP_OFF & _WDT_ON & _PWRTE_ON & _HS_OSC
```

```
OPTION_VAL equ B'01111000'    ;

#DEFINE capteur PORTB,1       ; capteur ligne 1
#DEFINE moteur PORTB,0        ; mise en service du moteur
#DEFINE LED PORTB,2           ; voyant d'alerte

    CBLOCK 0x00C              ; Zone de variables
    cmpt1 : 1                 ; compteur de boucles 1
    cmpt2 : 1                 ; compteur de boucles 2
    ENDC

; DEMARRAGE SUR RESET

    org 0x000                 ; Adresse de départ
    goto    init

; INITIALISATIONS

init bcf  STATUS,RP0          ; passer au bloc mémoire 0
    movlw     0xFF
    movwf     PORTB           ; initialiser le port B
    bsf       STATUS,RP0      ; passer au bloc mémoire 1
    movlw     OPTION_VAL      ; charger masque
    movwf     OPTION_REG      ; initialiser registre option
    movlw     0xFA            ; initialiser
    movwf     TRISB           ; les entrées sorties

    bcf STATUS,RP0            ; passer au bloc 0
    btfsc STATUS,NOT_TO       ; test du bit Time Out TO=0?
    goto start                ; sauter au programme principal
    bcf LED                   ; allumer le voyant d'alerte
    bcf moteur                ; arrêter le moteur
FIN
    goto FIN                  ; arrêt du système

; Sous-programme de temporisation de 6ms.

tempo
    movlw 7                   ; charger compteur2
    movwf cmpt2
boucle2
    clrf cmpt1                ; effacer compteur1
boucle1
    nop                       ; perdre 1 cycle
    decfsz cmpt1,f            ; décrémenter compteur1
    goto boucle1              ; si pas 0, boucler
    decfsz cmpt2,f            ; si 0, décrémenter compteur 2
```

```
    goto boucle2            ; si cmpt2 pas 0, recommencer boucle1
    return


;PROGRAMME PRINCIPAL

start
    btfsc capteur           ; lire le capteur RB1=0?
    goto start              ; si non activé attente
    clrwdt                  ; efface watchdog et TO=1
    call tempo              ; appeler la tempo de 6ms
    goto start              ; recommence lecture
    END                     ; fin d'assemblage
```

Chapitre 10
Le mode sommeil

10.1. Généralités

Le mode sommeil a pour but de diminuer la consommation du microcontrôleur par exemple quand il est alimenté sur pile ou sur batterie. Un composant PIC peut être mis en sommeil et ensuite réveillé par une interruption ou par le chien de garde.

En mode sommeil, l'oscillateur du microcontrôleur est arrêté, en conséquence le programme est stoppé ; la consommation du composant étant due principalement aux changements d'état simultanés des transistors (technologie CMOS) ; celle-ci est réduite au minimum. Toutefois l'utilisateur doit aussi faire en sorte que les connexions externes ne drainent pas trop de courant, notamment si elles sont en haute impédance (entrées flottantes).

10.2. Mise en œuvre

Le mode sommeil est provoqué par l'instruction SLEEP placée dans le programme.

A la mise en sommeil, le compteur du chien de garde est mis à zéro, mais continue de fonctionner (compteur indépendant), le bit TO (*Time Out*) du registre d'état est mis à 1, le bit PD (*Power Down*) du registre d'état est mis à 0, et l'oscillateur est arrêté.

Les ports d'entrées/sorties sont maintenus dans l'état.

10.2.1. *Le réveil*

Le réveil du microcontrôleur peut se faire :

– par un *reset*, dans ce cas l'utilisateur peut en examinant les bits TO et PD déterminer la cause du reset ;

– par un débordement du compteur du chien de garde si celui-ci est en service, dans ce cas il n'y a pas *reset* mais simplement un réveil qui permet l'exécution de l'instruction qui suit *sleep* ;

– par une demande d'interruption sur la ligne RB0/INT, sur une des lignes du port B ou à la fin de programmation de l'EEPROM si les masques correspondants ont été mis en place. Dans ce cas le bit GIE (General Interrupt Enable) n'a pas à être configuré.

Si GIE est à 0, les interruptions ne sont pas autorisées et le programme reprend à l'instruction qui suit l'instruction *sleep*.

Si GIE autorise les interruptions (à 1), le programme exécute l'instruction suivante puis se branche à l'adresse contenue à l'adresse 0x04, c'est à dire qu'il exécute le sous programme d'interruption. Dans ce cas on place un NOP après *sleep*.

EXEMPLE.– Programmer un octet dans l'EEPROM. Il suffit de mettre en oeuvre l'écriture et de la faire suivre par l'instruction *sleep,* dans ce cas le microcontrôleur est en sommeil et une fois l'écriture terminée (elle prend un temps assez long), le microcontrôleur sera réveillé par l'interruption de fin d'écriture.

10.2.2. *Le mode sommeil et les interruptions*

Le masque général GIE est à 0, et le masque et l'indicateur d'interruption sont mis à 1 :

– si l'interruption arrive avant l'instruction *sleep*, l'instruction *sleep* est traitée comme un NOP. Dans ce cas le chien de garde et son pré-diviseur ne sont pas remis à 0, le bit TO n'est pas mis à 1 et le bit PD n'est pas mis à 0 ;

– si l'interruption arrive pendant ou après l'instruction *sleep*, le microcontrôleur est réveillé immédiatement, l'instruction *sleep* est complètement exécutée avant le réveil, le chien de garde et son post-diviseur sont mis à 0, le bit TO (*Time Out*) est mis à 1 et le bit PD (*Power Down*) est mis à 0.

10.3. Exemples d'application

10.3.1. *Réalisation d'une temporisation*

On peut utiliser le mode sommeil avec le chien de garde pour réaliser une temporisation.

Si l'on place l'instruction SLEEP et que l'on a mis le chien garde en service, celui-ci pouvant facilement atteindre des durées jusqu'à plus de deux secondes, il réveillera le microcontrôleur et le programme reprendra après l'instruction SLEEP.

Si l'on règle le taux de division du post-diviseur du chien de garde sur 1/64 avec un quartz de fréquence 4 MHz, on va obtenir un réveil au bout d'environ une seconde. Dans le programme ci-dessous, la led va clignoter avec une période de deux secondes :

```
debut    bsf LED
         sleep
         bcf LED
         sleep
         goto debut
```

10.3.2. *Gestion d'un clavier*

Le mode sommeil peut être quitté par une interruption, et une interruption peut être générée par un changement d'état d'une des lignes du port B. Tout cela est parfaitement utilisable pour détecter l'appui sur une touche d'un clavier, de plus au repos en l'absence d'appui sur une touche le microcontrôleur sera en mode faible consommation.

Solution

On place les lignes RB0 à RB3 au niveau 0, les colonnes RB4 à RB7 en entrées puis le microcontrôleur en sommeil. L'appui sur une touche provoque le passage à 0 d'une colonne donc une interruption et le réveil du microcontrôleur.

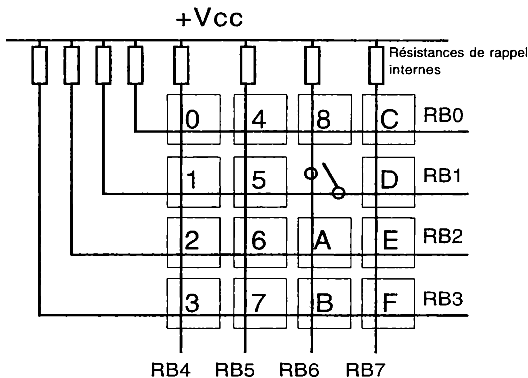


Figure 10.1. *Câblage du clavier sur le port B*

Le quartet de lecture des colonnes donnent la position horizontale de la touche.

On place ensuite les lignes en entrées et un niveau 0 sur les colonnes, le quartet de lecture des lignes donne la position verticale de la touche. Les deux lectures sont regroupées dans un seul octet. Il suffit alors de retrouver la valeur de la touche sachant que la touche 9 est à l'intersection de la ligne RB1 et de la colonne RB6.

La colonne RB4 a un poids de 0, la colonne RB5 un poids de 4, la colonne RB6 un poids de 8 et la colonne RB7 un poids de 12 avec les lignes ayant un poids respectif de 0 à 3.

Configurations à prévoir :

– mettre les résistances de rappel de potentiel du port B (RBPU à 1) ;

– mettre les lignes RB0 à RB3 en sortie à l'état bas ;

– autoriser les interruptions par changement d'état sur les lignes RB4 à RB7 (GIE et RBIE à 1) et mettre l'indicateur d'interruption correspondant RBIF à 1.

```
     list p=16F84
     #include <p16F84.inc>
OPTION_VAL equ B'10000000'   ; mettre les resistances du port B
INTMASK equ b'10001000'      ; autoriser les interruptions

     __CONFIG  _CP_OFF & _WDT_OFF & _PWRTE_OFF & _XT_OSC

     CBLOCK 0x0C
     COMPT : 1
     COMPT2 : 1
     COMPT3 : 1
     NB : 1
     LIGNE :1
     LECT : 1
     ENDC

     org 0X00
     goto INIT

; Procédure interruption
; l'interruption est provoquée par un changement d'etat sur une
; ligne RB4 a RB7.
; la lecture est recommencée 10 fois afin d'éliminer les aléas des
; rebonds de contact des touches

     org 0x004
RAZ  movlw 10             ; compteur pour l'antirebond
     movwf COMPT          ; dans COMPT
     call LECTURE
     movf LIGNE,W         ; charge la première lecture
     movwf LECT           ; mise en memoire dans LECT
LECT1 call LECTURE
     movf LIGNE,W         ; contenu de la lecture dans W
     xorwf LECT,W         ; compare la nouvelle lecture et l'ancienne
     btfss STATUS,Z       ; état de Z ? saut si Z=0
     goto LECT1           ; si different on recommence
```

```
LECT2 call LECTURE
     movf LIGNE,w
     xorwf LIGNE,w        ; compare les lectures
     btfss STATUS,Z       ; teste Z
     goto RAZ             ; recommence si different
     decf COMPT,f         ; decremente le compteur anti rebond
     btfsc STATUS,Z       ; si compteur à 0 conversion
     goto convers         ; la donnée est constante on la convertie
     bcf INTCON,RBIF      ; efface l'indicateur RBIF
     bcf STATUS,RP0       ; passe au bloc 0
     retfie


;initialisation et programme principal
;attente de l'appui sur une touche en mode sommeil

INIT bcf STATUS,RP0       ; passe au bloc mémoire 0
     movlw 0x00
     movwf PORTB          ; charge latch port B avec 00
     bsf STATUS,RP0       ; passe au bloc 1
     movlw 0xF0           ; poids forts en entrées =>niveau 1
     movwf TRISB          ; poids faibles en sortie => niveau 0
     movlw OPTION_VAL
     movwf OPTION_REG     ; programme registre OPTION
     bcf STATUS,RP0       ; passe au bloc 0
     movlw INTMASK
     movwf INTCON         ; autoriser les interruptions
     bcf INTCON,RBIF      ; raz du flag d'interutpion RBIF
     bcf STATUS,RP0       ; passe au bloc 0
     sleep                ; attente d'appui sur une touche
bou5 nop                  ; execute au retour d'interruption
     goto INIT            ; remise en attente


; Procedure lecture
; les lignes RB0 ‡ RB3 sont à l'état bas

LECTURE movf PORTB,W      ; lecture des colonnes
     andlw 0xf0           ; récupère poids fort
     movwf LIGNE          ; sauvegarde dans LIGNE
     bsf STATUS,RP0       ; passe au bloc mémoire 1
     swapf TRISB,f        ; inversion quartets entrées et sorties
     movlw 0x0F           ; charge 0F dans W
     bcf STATUS,RP0       ; passe au bloc 0
     andwf PORTB,f        ; mise à 0 du quartet poids fort
     movf PORTB,W         ; lecture des lignes
     andlw 0x0f           ; recupère poids faible
```

```
     addwf LIGNE,f         ; mise en mémoire dans LIGNE par addition
     bsf STATUS,RP0        ; passe au bloc 1
     swapf TRISB,f         ; inversion quartets entrée et sortie
     bcf STATUS,RP0        ; passe au bloc 0
     clrf PORTB            ; mise a 0 des lignes en sorties
     return

; sous programme de conversion de l'octet de lecture
; conversion quartet colonne.

convers   clrf COMPT2      ; mise à 0 du 2em compteur
     clrf NB
     comf LIGNE,f          ; complemente la lecture dans LIGNE
     btfss LIGNE,4         ; teste si bit 4 = 1
     clrf NB               ; ajoute 0 car 1ere colonne de touches
     btfsc LIGNE,5         ; teste si bit 5 = 1 (colonne poids 4)
     addlw 4               ;
     addwf NB,f            ; ajoute 4 a NB car 2eme colone
     clrw
     btfsc LIGNE,6         ; teste bit 6 (poids 8)
     addlw 8
     addwf NB,f            ; ajoute 8 a NB car 3eme colonne
     clrw
     btfsc LIGNE,7
     addlw 12
     addwf NB,f            ; ajoute 12 a NB car 4eme colonne
; conversion quartet lignes
     rrf LIGNE,f           ; décalage a droite avec bit C
     movlw 2               ; compteur de tests
     movwf COMPT3          ; dans COMPT3
bou  rrf LIGNE,f           ; décalage à droite
     btfss STATUS,C        ; teste bit C = 1 ?
     goto bou2             ; si C=0 on continue
     incf COMPT2,f         ; incrémente COMPT2
bou2 decf COMPT3,f         ; décrémente le compteur de tests
     btfss STATUS,Z        ; si Z=0 on boucle
     goto bou
     movf COMPT2,W         ; met COMPT2 dans W
     addwf NB,f            ; additionne NB + COMPT2 résultat dans NB
     return                ; valeur de la touche se trouve dans NB.

     END
```

Chapitre 11
Utilisation de l'EEPROM

L'EEPROM est une mémoire que l'on peut programmer et effacer électriquement, sans matériel spécifique. Le programme du microcontrôleur peut directement écrire des données dans cette mémoire non volatile, c'est-à-dire qu'elles seront mémorisées et conservées même en cas de coupure d'alimentation. L'effacement peut se faire par réécriture de données.

11.1. Mise en oeuvre

Le microcontrôleur PIC 16F84 contient 64 octets d'EEPROM, ces octets sont placés à l'adresse 2100h, mais cette adresse n'est pas accessible directement sauf pendant la programmation.

Dans ce cas on peut écrire :

```
ORG  0x2100    ; début de la zone d'EEPROM
DE 0x55        ; DE veut dite Data Eeprom
               ; suivi d'une donnée 0x55
```

Pour lire et écrire par logiciel, on utilise un adressage relatif grâce à un registre spécifique. L'adresse de début de la zone est 0x00 et l'adresse de fin 0x3F.

L'utilisation de l'EEPROM se fait grâce à quatre registres spéciaux :

– EEDATA, à l'adresse 0x08 du bloc mémoire 0, est un registre de transition pour les données en lecture et en écriture. C'est dans ce registre que transitent les données transférées ;

– EEADR, à l'adresse 0x09 du bloc mémoire 0, précise l'adresse de lecture ou d'écriture dans l'EEPROM. Ce registre est normalement capable d'adresser 256 octets, dans le cas du 16F84, il faut donc laisser ses deux bits de poids fort à 0 ;

EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD
88h	—	—	—	R/W -0	R/W -x	R/W -0	R/W -0	R/W -x

R : lecture, W : écriture, x : inconnu, u : inchangé, 1 : mis à 1 au reset, 0 : mis à 0 au reset

– EECON1, à l'adresse 0x88 du bloc 1, est le registre de contrôle d'accès à l'EEPROM, seuls 5 bits sont utilisés, les trois autres bits n'existent pas physiquement et sont lus comme des 0 ;

– EECON2, à l'adresse 0x89 du bloc mémoire 1, est un registre de contrôle qui n'existe pas physiquement, il ne sert qu'à des commandes spécifiques d'écriture. Sa lecture donne 0.

RD (*ReaD control bit*) : ce bit mis à 1 démarre un cycle de lecture. Il est remis à 0 automatiquement à la fin de la lecture. Il ne peut être mis à 0 par programme.

WR (*WRite control bit*) : ce bit démarre un cycle d'écriture quand il est mis à 1. Il est remis à 0 automatiqement à la fin du cycle d'écriture et ne peut être mis à 0 par logiciel.

WREN (*WRite ENable bit*) : ce bit autorise un cycle d'écriture quand il est mis à 1. A la mise en service du microcontrôleur (RESET), il est à 0.

WRERR (*WRite eeprom ERRor bit*) : c'est un indicateur d'erreur en écriture, il passe à 1 quand un cycle d'écriture a été interrompu par un RESET ou par le chien de garde. Il peut être lu par le programmeur pour éventuellement recommencer l'écriture. La donnée et l'adresse de travail restent inchangées.

EEIF (*EEprom write operation Interrupt Flag*) : ce bit passe à 1 quand l'écriture dans l'EEPROM est terminée, dans ce cas une demande d'interruption peut être demandée si elle est autorisée par le bit EEIE du registre INTCON (0Bh ou 8Bh).

11.2. Lecture dans l'EEPROM

Pour lire un octet il suffit de mettre une adresse dans le registre d'adresses EEADR (09h), puis de lancer le cycle de lecture avec le bit RD du registre EECON1. Le résultat de la lecture se trouve alors dans le registre de données EEDATA jusqu'à ce qu'elle soit modifiée par le programmeur (nouveau cycle de lecture ou d'écriture).

EXEMPLE.– On veut lire l'octet situé à l'adresse 0x03 dans l'EEPROM.

```
bcf STATUS,RP0      ; passe au bloc mémoire 0
movlw 0x03          ; charge l'adresse de lecture
movwf EEADR         ; dans le registre d'adresse
bsf STATUS,RP0      ; passe au bloc 1
bsf EECON1,RD       ; lance le cycle de lecture
bcf STATUS,RP0      ; passe au bloc 0
movf EEDATA,W       ; met la donnée dans l'accumulateur W
```

Ce cycle de lecture assez long est souvent utilisé, il est donc intéressant de le transformer en macro-instruction pour simplifier le travail du programmeur. Cette macro pouvant être placée dans le fichier “include”, il suffit alors de prévoir le passage de paramètre.

EXEMPLE.– Définir une macro avec passage de paramètre ; dans le cas de la lecture le paramètre est évidemment l'adresse de lecture, le résultat de la macro se trouvant au retour dans l'accumulateur W.

```
LECTEE macro ADREE      ; ADREE est le paramètre
     movlw ADREE        ; adresse de l'octet à lire
     movwf EEADR        ; mettre l'adresse dans le registre
     bsf STATUS,RP0     ; passer au bloc 1
     bsf EECON1,RD      ; lancer la lecture
     bcf STATUS,RP0     ; passer au bloc 0
     movf EEDATA,W      ; lecture dela donnée
     endm               ; fin de la macro
```

Pour utiliser cette macro que l'on pourra placer dans le fichier include, il suffit d'écrire :

```
     LECTEE 0xXX     ; 0xXX étant l'adresse de la donnée à lire en
                     ; EEPROM.
```

11.3. Ecriture dans l'EEPROM

Pour écrire il suffit de mettre l'adresse de la destination dans le registre EEADR et l'octet à écrire dans le registre EEDATA, puis de lancer l'opération d'écriture par la mise à 1 du bit WR du registre EECON1.

L'utilisateur doit toutefois suivre une procédure particulière imposée par le constructeur pour initialiser cette écriture :

```
     bsf STATUS,RP0     ; passer au bloc 1
     bcf INTCON,GIE     ; masquer les interruptions
     bsf EECON1,WREN    ; autoriser l'écriture
; lignes obligatoires
     movlw 0x55
     movwf EECON2       ; écrire 55h
     movlw 0xAA
     movwf EECON2       ; écrire AAh
     bsf EECON1,WR      ; lancer l'écriture
     bsf INTCON,GIE     ; autorise les interruptions éventuellement
```

A la fin du cycle d'écriture, il s'écoule encore 10 ms avant que la donnée ne soit écrite en mémoire. Il faut donc attendre et vérifier que tout soit terminé avant de recommencer un autre cycle ou de lire cette donnée. La fin de l'écriture peut être détectée en lisant l'état de l'indicateur EEIF, ou encore en lisant l'état du bit WR qui est remis automatiquement à 0 en fin de cycle.

Le bit WREN n'est pas remis à 0 automatiquement en fin d'écriture. Après la fin du cycle d'écriture le bit EEIF passe à 1, ce qui permet éventuellement de demander une interruption si le bit EEIE le permet.

Il est intéressant de créer une macro pour l'écriture, celle-ci devra donc passer deux paramètres : l'adresse et la donnée. La macro qui suit doit être appelée après avoir mis la donnée dans l'accumulateur W, le paramètre ADRECRI est l'adresse d'écriture.

Il est évident qu'il faut interdire les interruptions pendant un cycle d'écriture.

```
ECRIEE macro ADRECRI      ; adrecri est le paramètre
     bcf STATUS,RP0       ; bloc mémoire 0
     movwf EEDATA         ; la donnée est placée dans le registre
movlw ADRECRI             ; l'adresse est chargée
     movwf EEADR          ; dans le registre d'adresses
     bcf INTCON,GIE       ; interdire les interruptions
     bsf STATUS,RP0       ; passer au bloc 1
     bcf EECON1,EEIF      ; mettre à 0 l'indicateur
     bsf EECON1,WREN      ; autoriser l'écriture
     movlw 0x55
     movwf EECON2
     movlw 0xAA
     movwf EECON2
     bsf EECON1,WR        ; lancer l'écriture
     bcf EECON1,WREN      ; empêcher une prochaine écriture
     bcf STATUS,RP0       ; repasser au bloc 0
     endm
```

11.4. Vérification de l'écriture

Dans des EEPROM très souvent programmées parfois à la limite du nombre maximum prévu par le constructeur, il peut être utile de vérifier l'information écrite.

Exemple de vérification d'écriture :

```
     bcf STATUS,RP0       ; passer au bloc 0
     movf EEDATA,W        ; mettre la donnée dans W
     bsf STATUS,RP0       ; passer au bloc 1
LECTURE
```

```
     bsf EECON1,RD         ; déclencher la lecture : RD à 1
     bcf STATUS,RP0        ; passer au bloc 0
;comparaison entre la donnée lue et la donnée prévue (dans W)
     subwf EEDATA,W        ; soustraction pour comparaison
     btfss STATUS,Z        ; test du bit Z = 0 ?
     goto ERREUR           ; si Z=0 continue le programme
                           ; si Z=1 on saute à un s/prog d'erreur
```

11.5. L'interruption de fin d'écriture

La fin d'un cycle d'écriture peut être la source d'une demande d'interruption. Pour cela il faut autoriser cette interruption en mettant le masque EEIE du registre INTCON (0Bh) à 1 et le masque général des interruptions GIE à 1. Cette interruption ne peut survenir qu'après le cycle d'écriture.

11.6. Application

L'exercice qui suit n'est qu'un exercice de démonstation afin de vérifier le bon fonctionnement de la lecture et de l'écriture dans l'EEPROM. Il utilise une macro pour l'écriture.

Il réalise la lecture d'une roue codeuse branchée sur le port A, enregistre le chiffre lu dans l'EEPROM, vérifie l'écriture puis affiche le chiffre écrit sur le port B si tout s'est bien passé sinon affiche 0xFF.

```
     list P=16F84
     __CONFIG _CP_OFF & _WDT_OFF & _XT_OSC
     include <p16f84.inc>

     CBLOCK 0x0c
     chiffre : 1
     ENDC

ADRECRI equ 0x15           ; adresse d'écriture

ECRIEE macro ADRECRI       ;
     bcf STATUS,RP0        ; bloc mémoire 0
     movwf EEDATA          ; donnée dans le registre de donnée
     movlw ADRECRI         ; l'adresse est chargée
     movwf EEADR           ; dans le registre d'adresses

     bsf STATUS,RP0        ; passer au bloc 1
     bcf INTCON,GIE        ; interdire les interruptions
     bcf EECON1,EEIF       ; mettre à 0 l'indicateur
```

```
     bsf EECON1,WREN     ; autoriser l'écriture
     movlw 0x55
     movwf EECON2
     movlw 0xAA
     movwf EECON2
     bsf EECON1,WR       ; lancer l'écriture
     bcf EECON1,WREN     ; empêcher une prochaine écriture
     endm

     org 0x00
     goto INIT

;initialisation et programme principal

INIT bcf STATUS,RP0      ; passe en bloc 0
     movlw 0x00
     movwf PORTB         ; charge latch port B avec 00
     movwf PORTA
     bsf STATUS,RP0      ; passe en bloc 1
     movlw 0xff          ; poids forts en entrée =>niveau 1
     movwf TRISA         ; poids faible en sortie => niveau 0
     movlw 0x00          ; port A en entrée
     movwf TRISB         ; port B en sortie

; écriture en EEPROM à l'adresse ADRECRI
     bcf STATUS,RP0
     movf PORTA,W        ; lecture du port A
     movwf chiffre       ; sauvegarde dans chiffre
     ECRIEE ADRECRI      ; lancement de la macro

; attendre la fin de l'écriture
attend btfss EECON1,EEIF     ; le flag est-il passé à 1 ?
     goto attend             ; si non attendre, si oui verif

; Procedure de vérification

verif bcf STATUS,RP0
     movf chiffre,w      ; reprendre la donnée à écrire
lecture bsf STATUS,RP0   ; bloc 1
     bsf EECON1,RD       ; lancer la lecture
     bcf STATUS,RP0      ; bloc 0
; comparaison et affichage
     subwf EEDATA,f      ; soustraction
     bcf STATUS,RP0      ; bloc 0
     btfss STATUS,Z      ; valeurs identiques ? Z=1 ?
     goto erreur
     movf chiffre,W      ; reprend le chiffre
```

```
     movwf PORTB            ; l'écrit sur le portB
Fin  goto Fin
erreur    movlw 0XFF        ; si erreur
     movwf PORTB            ; écriture de FF sur port B
     goto Fin
     END
```

Chapitre 12
Le PIC 16F877

Le microcontrôleur PIC 16F877 possède la même structure de base, les mêmes composants techniques, le même jeu d'instructions (35) et les mêmes modes d'adressage que le PIC 16F84, mais dispose de nombreuses interfaces supplémentaires qui font de lui l'un des plus performants et des plus complets de la gamme.

En technologie CMOS FLASH, la vitesse maximum du quartz de l'horloge atteint 20 MHz pour certains modèles de PIC et la tension d'alimentation peut varier de 2 volts à 5,5 volts. Différent du PIC 16F84, il possède 14 niveaux d'interruption dus aux nombreuses interfaces, les possibilités d'être programmé avec la seule tension de cinq volts (sans Vpp) et d'être « debuggé » *via* deux de ses connexions (RB6 et RB7).

12.1. Les interfaces

Il comprend :

– 5 ports parallèles ;

– un compteur temporisateur TMR0 sur 8 bits avec prédiviseur ;

– un compteur temporisateur TMR1 sur 16 bits avec prédiviseur, pouvant être incrémenté par une horloge externe ;

– un compteur temporisateur TMR2 sur 8 bits avec un registre de période sur 8 bits, un prédiviseur et un post-diviseur ;

– un module de capture sur 16 bits ;

– un module de comparaison sur 16 bits ;

– un générateur d'impulsions modulées en largeur sur 10 bits (PWM) ;

– un port série MSSP comprenant une liaison série synchrone maître/esclave SPI et une liaison série synchrone maître/esclave I2C ;

– une liaison série synchrone/asynchrone émetteur/récepteur avec possibilité de détection d'adresse sur 9 bits ;

– un port parallèle esclave sur 8 bits avec des lignes de contrôle RD, WR et CS ;

– 8 entrées de convertisseurs analogiques numériques 10 bits ;

– un circuit de détection *Brown-Out Reset* (BOR) pour surveiller la tension d'alimentation.

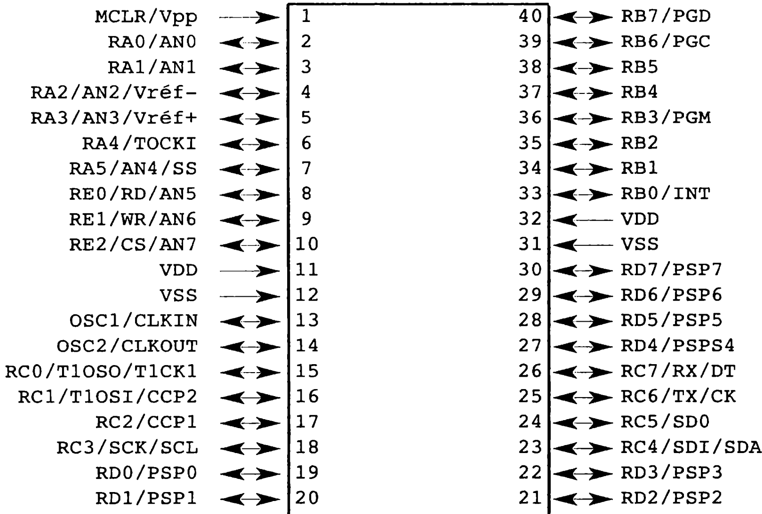


Figure 12.1. *Connexions du PIC 16F877*

Le composant ne comprenant que 40 connexions, il est évident que celles-ci sont communes à plusieurs interfaces.

Ce qui suit permet de découvrir toutes ces interfaces sans pour autant les étudier en détail, les deux chapitres suivants permettront d'étudier les deux interfaces de communication série synchrone car elles sont de plus en plus employées en électronique.

12.2. Les mémoires

12.2.1. *La mémoire de programme*

Sa mémoire programme contient 8 K emplacements de mémoire FLASH sur 14 bits. L'adressage de ces 8 K nécessite 13 bits d'adresse. Il faut donc utiliser en plus des 8 bits du PCL, les cinq bits prévus dans PCLATH. Pour atteindre lors de saut ou de branchement tous les emplacements de cette mémoire, l'opérande de l'instruction étant sur 11 bits il est nécessaire de programmer les bits 3 et 4 du registre PCLATH.

Mémoire RAM du 16F877							
BLOC 1		BLOC 2		BLOC 3		BLOC 4	
00h	addr. indirect	80h	addr. indirect	100h	addr. indirect	180h	addr. indirect
01h	TMR0	81h	OPTION_REG	101h	TMR0	181h	OPTION_REG
02h	PCL	82h	PCL	102h	PCL	182h	PCL
03h	STATUS	83h	STATUS	103h	STATUS	183h	STATUS
04h	FSR	84h	FSR	104h	FSR	184h	FSR
05h	PORTA	85h	TRISA	105h		185h	
06h	PORTB	86h	TRISB	106h	PORTB	186h	TRISB
07h	PORTC	87h	TRISC	107h		187h	
08h	PORTD	88h	TRISD	108h		188h	
09h	PORTE	89h	TRISE	109h		189h	
0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah	PCLATH
0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh	INTCON
0Ch	PIR1	8Ch	PIE1	10Ch	EEDATA	18Ch	EECON1
0Dh	PIR2	8Dh	PIE2	10Dh	EEADR	18Dh	EECON2
0Eh	TMR1L	8Eh	PCON	10Eh	EEDATH	18Eh	Réservé
0Fh	TMR1H	8Fh		10Fh	EEADRH	18Fh	Réservé
10h	T1CON	90h		110h	Registres à usage général 16 octets	190h	Registres à usage général 16 octets
11h	TMR2	91h	SSPCON2	111h		191h	
12h	T2CON	92h	PR2	112h		192h	
13h	SSPBUF	93h	SSPADD	113h		193h	
14h	SSPCON	94h	SSPSTAT	114h		194h	
15h	CCPR1L	95h		115h		195h	
16h	CCPR1H	96h		116h		196h	
17h	CCP1CON	97h		117h		197h	
18h	RCSTA	98h	TXSTA	118h		198h	
19h	TXREG	99h	SPBRG	119h		199h	
1Ah	RCREG	9Ah		11Ah		19Ah	
1Bh	CCPR2L	9Bh		11Bh		19Bh	
1Ch	CCPR2H	9Ch		11Ch		19Ch	
1Dh	CCP2CON	9Dh		11Dh		19Dh	
1Eh	ADRESH	9Eh	ADRESL	11Eh		19Eh	
1Fh	ADCON0	9Fh	ADCON1	11Fh		19Fh	
20h	Registres à usage général 96 octets	A0h	Registres à usage général 80 octets	120h	Registres à usage général 80 octets	1A0h	Registres à usage général 80 octets
		EFh		16Fh		1Eh	
		F0h	Accès 70h - 7Fh	170h	Accès 70h - 7Fh	1F0h	Accès 70h - 7Fh
7Fh		FFh		17Fh		1FFh	

Figure 12.2. *Les registres du 16F877*

12.2.2. *La mémoire RAM, ou mémoire de données*

La RAM (FILE) du 16F877 de 512 octets, contenant les registres spéciaux et les registres à usage général, est divisée en quatre blocs mémoire. Ces quatre blocs sont sélectionnés grâce aux bits RP0 et RP1 du registre d'état (STATUS).

Pour l'adressage indirect, les 512 octets nécessitant 9 bits d'adressage, la sélection de l'adresse dans l'un des quatre blocs se fait en utilisant le bit IRP du registre d'état en complément du registre d'adressage indirect FSR.

12.2.3. *La mémoire EEPROM*

Accessible en lecture et en écriture par une procédure spéciale, elle contient 256 octets adressables de 00h à FFh.

12.2.4. *La pile*

La pile est constituée de 8 registres de 13 bits situés en dehors de la mémoire de données et en dehors de la mémoire de programme.

Le pointeur de pile ne peut être ni lu ni écrit. Le PCLATH n'est pas modifié lors du chargement ou du déchargement de la Pile.

12.3. Les ports parallèles

12.3.1. *Le port A*

Ce port situé à l'adresse 05h dans le bloc mémoire 00 comprend six connexions programmables en entrées/sorties grâce au registre TRISA à l'adresse 85h du bloc mémoire 01. Son fonctionnement est identique à celui du PIC16F84.

Ses connexions sont communes avec les entrées des convertisseurs A/N, l'entrée de compteur TOCKI (RA4) et RA5 est aussi utilisée par la liaison SPI (signal SS).

12.3.2. *Le port B*

Situé aux adresses 06h ou 106h, son fonctionnement est identique à celui du PIC16F84. Ses entrées/sorties dépendent du registre TRISB aux adresses 86h ou 186h. La ligne RB0 peut être utilisée comme entrée de demande d'interruption (INT), la ligne RB3 (PGM) permet de commander le mode de programmation sans Vpp et les lignes RB6 et RB7 servent aussi respectivement de ligne d'horloge et de donnée pour la programmation du microcontrôleur et l'émulateur.

12.3.3. *Le port C*

Le port C à l'adresse 07h comprend huit lignes d'entrées/sorties dont le sens dépend du registre TRISC à l'adresse 17h.

Les lignes RC3, RC4, RC5 peuvent être utilisées pour la liaison série synchrone SPI, respectivement pour SCK, SDI, SDO et les lignes RC3 et RC4 pour la liaison série synchrone I2C respectivement SCL, SDA.

Les lignes RC6 et RC7 sont communes avec la liaison série asynchrone USART respectivement RX et TX, et aussi avec cette liaison en mode synchrone respectivement CK (horloge) et DT (donnée).

La ligne RC0 est utilisable comme sortie d'oscillateur du compteur TMR1 (T1OSO) ou comme entrée d'horloge du TMR1(T1CKI), la ligne RC1 comme entrée d'oscillateur du TMR1 (T1OSI). La ligne RC1/CCP2 comme entrée de capture 2, sortie de capture 2 ou sortie de générateur d'impulsion modulée en largeur PWM2 et la ligne RC2/CCP1 comme entrée de capture 1, sortie de capture 1 ou sortie de générateur d'impulsion modulée en largeur PWM1.

12.3.4. *Le port D*

Le port D à l'adresse 08h comprend huit lignes d'entrées/sorties dont le sens dépend du registre TRISD à l'adresse 88h.

Ses huit lignes (PSP0 à PSP7) peuvent aussi servir comme port parallèle dit « esclave » dont les lignes de commandes associées : RD pour la lecture, WR pour l'écriture et CS pour la sélection de boitier sont réalisées avec les lignes du port E.

12.3.5. *Le port E*

Ce port est composé de trois lignes bi-directionnelles dont le sens est déterminé par le registre TRISE à l'adresse 89h.

Ces trois lignes constituent aussi les trois entrées de convertisseurs AN5 à AN7 et les lignes de commandes du bus esclave PSP (RD, WR, CS).

12.4. Le compteur temporisateur TMR0

Situé à l'adresse 01h ou 101h, son fonctionnement est identique au TMR0 du PIC 16F84.

12.5. Le compteur temporisateur TMR1

Le TMR1 peut avoir deux usages : compteur ou temporisateur, il est composé de deux registres 8 bits accessibles en lecture et en écriture TMR1L (0Eh) et TMR1H (0Fh).

Le choix du mode fonctionnement compteur ou temporisateur se fait par le bit TMR1CS du registre T1CON, ce bit à 0 détermine un comptage interne donc le mode temporisateur.

12.5.1. *Mode temporisateur*

Ce mode est mis en service par le bit TMR1ON du registre T1CON.

Le compteur du temporisateur est incrémenté par FOSC/4, il passe donc de 0 à FFFFh puis repasse à 0.

Le passage à 0 fait passer le bit TMR1IF du registre PIR0 à 1, ce qui peut déclencher une interruption si elle autorisée par le bit TMR1IE du registre PIE0.

L'oscillateur de TMR1 est mis en service par la mise à 1 du bit TOSCEN.

Les lignes RC1/TIOSC0 et RC1 (TIOSC0) sont en entrées pour réaliser l'oscillateur de l'horloge externe et les bits de direction TRISC(0) et TRISC(1) sont ignorés.

12.5.2. *Mode compteur*

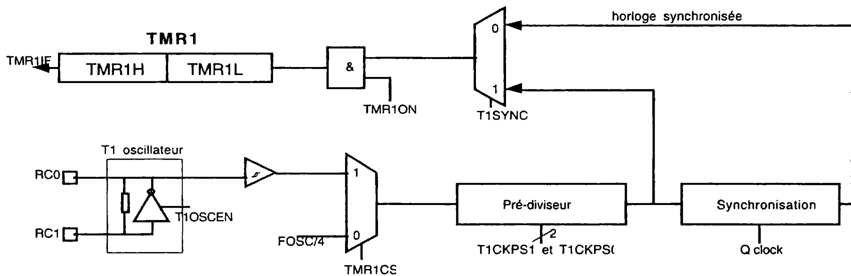


Figure 12.3. *Structure du compteur TMR1*

Le compteur peut fonctionner en mode synchrone ou en mode asynchrone.

Il est incrémenté par chaque front montant sur l'entrée RC1/T1OSI quand le bit T1OSCEN est à 1 ou sur la connexion RC0/T1OSO si le bit T1OSCEN du registre T1CON est à 0.

Si le bit T1SYNC est à 0, l'entrée d'horloge est synchronisée sur l'horloge interne après la pré-division.

Si le bit T1SYNC est à 1, l'horloge externe n'est pas synchronisée et peut continuer à compter pendant le sommeil du microcontrôleur, le débordement du compteur peut alors réveiller celui-ci.

12.6. Le timer TMR2

Le temporisateur TMR2 est composé d'un compteur 8 bits avec prédiviseur et post-diviseur.

Il peut être utilisé comme base de temps du générateur d'impulsion modulée en largeur des modules CCP (Capture/Compare/PWM).

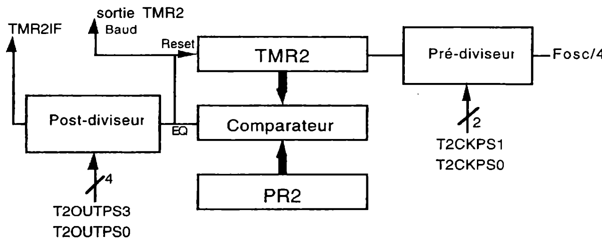


Figure 12.4. *Structure du compteur TMR2*

Ce module possède un registre de période PR2 auquel il est comparé pour générer la modulation de largeur d'impulsion.

La fréquence FOSC/4 utilisée comme base de temps de comptage peut être prédivisée selon l'état des bits T2CKPS0 et T2CKPS1 du registre de contrôle du timer 2 T2CON. Le registre TMR2 s'incrémente à partir de 0 jusqu'à égalité avec le contenu de PR2, ce qui le fait repasser à 0. La sortie de TMR2 peut aussi être utilisée en générateur de bauds pour réaliser les durées de la liaison série.

Le signal de sortie du module TMR2 (sortie du comparateur) peut être post-divisé par un taux de 1 à 16 selon l'état des bits T2OUTPS0 à T2OUTPS3 du registre T2CON.

Le registre PR2 est initialisé avec FFh au moment du reset.

La sortie du post-diviseur fait passer l'indicateur TMR2IF du registre PIR1 à 1 ce qui peut déclencher une interruption si le bit TMR2IE du registre PIE2 l'autorise.

12.7. USART ou *universal synchronous asynchronous receiver transmitter*

Cette interface est aussi appelée SCI (*Serial Communication Interface*).

Elle peut fonctionner en *half duplex* en mode synchrone maître ou esclave, ou en *full duplex* en mode asynchrone.

Le bit SPEN du registre RCSTA et les bits 6 à 7 du registre TRISC permettent de configurer les connexions RC6 (Tx) et RC 7 (Rx).

12.7.1. *Mode asynchrone*

Ce mode est sélectionné par la mise à 0 du bit SYNC dans le registre TXSTA.

Le format des données transmises est de type NRZ (non retour à 0), de longueur 8 ou 9 bits avec bit de stop mais sans bit de parité (celui-ci peut être éventuellement créé par le programme de transmission).

12.7.1.1. *Transmission asynchrone*

Un registre à décalage transmet en série les mots issus du registre de transmission TXREG, la vitesse de transmission est définie par un générateur de bauds SPBRG. Le programmeur a la possibilité de transmettre un 9e bit par la mise à 1 du bit TX9 du registre TXSTA.

A la fin du transfert une interruption peut être demandée par le bit indicateur de fin de transmission TXIF du registre PIR1 si le bit TXIE du registre PIEl l'autorise et si les bits PEIE et GIE sont bien configurés.

12.7.1.2. *Réception asynchrone*

La réception est mise en service par le bit CREN du registre RCSTA.

L'information série charge le registre à décalage RSR par l'entrée RX, puis celui-ci est transféré dans le registre RCREG après la détection du bit de STOP. La réception est autorisée par la mise à 0 du bit CREN du registre RCSTA et la ligne Rx est contrôlée par le bit SPEN du registre RCSTA.

Quand la réception est terminée, le bit RCIF du registre PIR1 est mis à 1 ce qui peut déclencher une interruption si le bit RCIE du registre PIE1 l'autorise.

Si des erreurs sont détectées, les bits OERR (*Overrun Error bit*) et FERR (*Framing Error bit*) du registre RSCSTA sont mis à 1.

Mode opératoire :

– choisir la vitesse en initialisant le registre SPBRG ;
– mettre à 0 le bit SYNC ;
– mettre le port en service avec le bit SPEN ;
– mettre RCIE à 1 pour autoriser les interruptions ;
– mettre RX9 à 1 pour utiliser le 9e bit ;
– mettre CREN à 1 pour la réception ;
– RCIF passe à 1 quand la réception est terminée dans ce cas une interruption peut être déclenchée si elle est autorisée ;
– lire le registre RCSTA pour obtenir le 9e bit et connaître les éventuelles erreurs ;
– lire les 8 bits de données dans RCREG ;
– mettre CREN à 0 pour effacer les indicateurs d'erreur ;
– vérifier que GIE et PEIE sont à 1.

12.7.2. *Mode synchrone*

Dans ce cas l'interface fonctionne en half duplex, elle est configurée par la mise à 1 du bit SYNC du registre TXSTA et par la mise à 1 du bit SPEN du registre RCSTA. Deux lignes sont utilisées : la ligne RC6 donne le signal CK (horloge) et la ligne RC7 le signal DT (donnée).

12.7.2.1. *Transmission du maître*

Le mode maître est déterminé par la mise à 1 du bit CSRC du registre TXSTA.

La transmission est mise en service par le bit TXEN du registre TXSTA mais une transmission ne peut commencer si TXREG n'est pas chargé. Le premier bit est sorti sur le front montant de la ligne d'horloge CK. La mise à 1 de TXEN provoque le démarrage du générateur de baud et crée le premier bit d'horloge.

L'émission consiste au transfert du registre de transmission TXREG dans le registre à décalage TSR. Quand le registre de transmission est vide le bit TXIF du registre PIR1 passe à 1 ce qui peut déclencher une interruption si elle est autorisée par le bit TXIE du registre PIE1 à 1.

Le registre TSR est inaccessible mais le bit TRMT du registre TXSTA indique, quand il est à 1, que le registre TSR est vide.

Mode opératoire :
- initialiser le générateur de bauds ;
- mettre en service la transmission avec SYNC, SPEN et CSRC à 1 ;
- mettre éventuellement TXIE à 1 pour l'interruption ;
- mettre éventuellement TX9 à 1 pour l'usage du 9e bit ;
- mettre à 1 le bit TXEN ;
- charger le 9e bit dans TX9D si nécessaire ;
- charger le registre TXREG avec la donnée pour démarrer la transmission ;
- vérifier la mise à 1 des bits GIE et PEIE.

12.7.2.2. *Réception par le maître*

Le mode synchrone étant sélectionné, la réception est commandée par la mise à 1 du bit SREN du registre RCSTA. La donnée est échantillonnée par le front descendant de l'horloge. Après le dernier bit reçu, le registre à décalage est transféré dans le registre RCREG et le bit RCIF de PIR1 passe à 1, ce qui peut provoquer une interruption si RCIE de PIE1 l'autorise. Le bit RCIF repasse à 0 quand le registre RCREG est vidé. Pour le 9e bit, il faut lire auparavant le bit RX9D dans le registre RCSTA.

Mode opératoire :

– initialiser le registre de bauds ;

– mettre les bits SYNC, SPEN et CSRC à 1 ;

– vérifier si CREN et SREN sont à 1 ;

– mettre éventuellement RCIE à 1 pour autoriser l'interruption ;

– mettre éventuellement RX9 à 1 pour gérer le 9e bit ;

– mettre SREN à 1 pour réception d'un octet ou CREN à 1 pour une réception continue ;

– le bit RCIF passe à 1 quand la réception est terminée, ce qui peut provoquer une interruption ;

– lire le registre RCSTA pour obtenir le 9e bit et les bits d'erreur ;

– lire RCREG pour obtenir la donnée ;

– vérifier si les bits GIE et PEIE sont à 1.

12.7.2.3. *Transmission par l'esclave*

Le signal d'horloge, qui permet la transmission, arrive du maître par la ligne RC6/CK. Le mode esclave est sélectionné par la mise à 1 du bit CSRC du registre TXSTA.

Mode opératoire :

– les bits SYNC et SPEN sont mis à 1 et le bit CSRC à 0 ;

– CREN et SREN sont mis à 0 ;

– TXIE peut être mis éventuellement à 1 pour l'interruption ;

– TX9 peut être mis éventuellement à 1 pour utiliser le 9e bit ;

– TXEN est mis à 1 pour autoriser la transmission ;

– mettre le 9e bit dans TX9D ;

– lancer la transmission en chargeant le registre TXREG ;

– vérifier l'état des bits GIE et PEIE.

12.7.2.4. *Réception par l'esclave*

En mode sommeil pour l'esclave, le bit SREN est inutile.

Mode opératoire :

– mettre SYNC et SPEN à 1 et CSRC à 0 ;

– RCIE peut éventuellement être mis à 1 pour l'interruption ;

– RX9 peut éventuellement être mis à 1 pour le 9e bit ;

– la réception est mise en service par le bit CREN à 1 ;

– RCIF passe à 1 quand la réception est terminée ce qui peut déclencher une interruption ;

– lire le registre RCSTA pour le 9e bit et les erreurs ;
– lire le registre RCREG pour obtenir la donnée ;
– remettre à 0 le bit CREN en cas d'erreur ;
– vérifier l'état des bits GIE et PEIE.

12.8. Le module convertisseur

Le module convertisseur dispose de huit entrées analogiques, dont deux entrées de tension de référence. Il procède par approximations successives, donne les résultats sur 10 bits et peut aussi fonctionner pendant le mode sommeil car il dispose d'un oscillateur RC interne.

Il utilise quatre registres :
– ADRESH et ADRESL pour contenir les résultats sur 10 bits ;
– ADCON0 et ADCON1 pour contrôler le fonctionnement.

C'est le bit GO/DONE à 1 qui met la conversion en route.

Une fois la conversion terminée, le résultat est dans la paire de registres, le bit GO/DONE du registre ADCON0 passe à 0, le bit ADIF, indicateur de fin de conversion, du registre PIR1 passe à 1 et une interruption peut être déclenchée si le bit ADIE de PIE1 l'autorise. Le bit ADFM du registre ADCON1 permet de justifier le résultat dans les registres ADRESH et ADRESL. Si ADFM est à 1, le résultat est justifié à droite donc les 6 bits de poids forts du résultat sont mis à 0 et inversement.

ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	-	ADON
1Fh	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0

Quand le module est configuré, le canal doit être acquis avant de commencer la conversion. Les lignes doivent être configurées en entrées.

Mode opératoire :

– configurer les lignes analogiques et les références grâce aux bits PCFG0 à PCFG3 du registre ADCON1 ;

– sélectionner le canal d'acquisition avec les bits CHS0 à CHS2 ;

– sélectionner la vitesse de l'horloge de conversion avec les bits ADCS0 à ADCS1 du registre ADCON0 ;

– mettre en service le module avec le bit ADON à 1 du registre ADCON0 ;

– configurer éventuellement les interruptions avec les bits ADIF à 0 du registre PIR1, ADIE à 1 du registre PIE1, et les bits généraux PEIE à 1 et GIE à 1 ;

– attendre le temps d'acquisition ;

– démarrer la conversion avec GO/DONE à 1 ;

– attendre la fin de la conversion, c'est-à-dire que GO/DONE passe à 0 ou qu'une interruption soit déclenchée ;

– lire le résultat de la conversion dans la paire de registres ; ADRESH puis ADRESL ;

– mettre à 0 le bit ADIF du registre PIR1 si nécessaire.

ADCON1	ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0
14h	U-0	U-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0

12.9. Modules Capture, comparaison, et modulation de largeur d'impulsion

Chaque module CCP (*Capture-Compare-Pulse*) contient un registre 16 bits qui peut être utilisé selon trois modes de fonctionnement.

Le module CCP1, constitué deux registres 8 bits CCPR1L et CCPR1H, est contrôlé par le registre CCP1CON. Son fonctionnement est associé à TMR1. Il pilote la connexion CCP1/RC1. Le déclenchement spécial provoque la remise à 0 de PMR1.

Le module CCP2 est constitué de deux registres 8 bits CCPR2L et CCPR2H, il est contrôlé par le registre CCP2CON et son fonctionnement associé à TMR1 est identique à celui de CCP1.

Le déclenchement spécial provoque la remise à 0 de TMR1 mais peut aussi lancer une conversion analogique numérique si elle est en service. La connexion pilotée est CCP2/RC2.

12.9.1. *Mode capture*

Ce mode utilise le compteur TMR1.

Le registre du module capture le contenu du registre TMR1 quand un événement apparaît sur la connexion RCx/CCPx qui doit alors être programmée en entrée.

La caractéristique de cet événement (signal) est déterminée par les bits CCPxM0 à CCPxM3 du registre CCPxCON; la capture peut avoir lieu :

– à chaque front descendant ;

– à chaque front montant ;

– tous les quatre fronts montants ;

– tous les 16 fronts montants.

Quand une capture est réalisée, le bit indicateur CCP1IF du registre PIR1 passe à 1 ce qui provoquer une interruption si elle est autorisée par le bit CCP1IE du registre PIE1. L'indicateur CCP1IF doit être remis à 0 par le logiciel.

12.9.2. *Mode Compare*

Ce mode utilise le compteur TMR1.

Le registre du module est comparé en permanence avec le contenu de TMR1, et quand il y a égalité, la connexion RCx/CCPx, qui doit être configurée en sortie, peut générer les signaux suivants :

- passage au niveau 1 et l'indicateur CCP1IF passe à 1;
- passage au niveau 0 et l'indicateur CCP1IF passe à 1;
- reste inchangé mais une interruption est déclenchée (CCP1GF passe à 1).

Le signal généré dépend de la programmation des bits CCPxM0 à CCPxM3 du registre CCPxCON.

12.9.3. *La modulation de largeur d'impulsion*

Cet interface utilise le compteur TMR2.

Le fonctionnement est déterminé par les bits CCPxM0 à CCPxM3 du registre CCPxCON.

Ce module permet la génération de signaux rectangulaires de fréquence et de rapport cyclique variables. Pour cela il utilise deux registres PR2 et CCPRxL avec le compteur TMR2.

La période du signal est déterminée par le contenu du registre PR2 et le rapport cyclique par le contenu du registre CCPRxL auquel on ajoute deux bits afin de réaliser une résolution sur 10 bits. Le signal généré est obtenu sur la connexion CCPx qui doit être configurée en sortie.

La partie du signal au niveau 1 est appelée « duty cycle ».

12.9.3.1. *Fonctionnement du module*

Quand le contenu du registre TMR2 est égal au contenu du registre PR2 :

- TMR2 est mis à 0 ;
- la connexion CCPx passe à 1 ;
- le contenu du registre CCPRxL est transféré dans le registre CCPRxH.

Le comptage démarre et le compteur TMR2 est incrémenté et est comparé en permanence avec le registre CCPRxH.

Quand TMR2 est égal à CCPRxH :

- la connexion CCP1 passe à 0.

Le comptage continue et TMR2 est comparé en permanence avec le registre PR2.

Quand TMR2 est égal à PR2 :

– le cycle recommence.

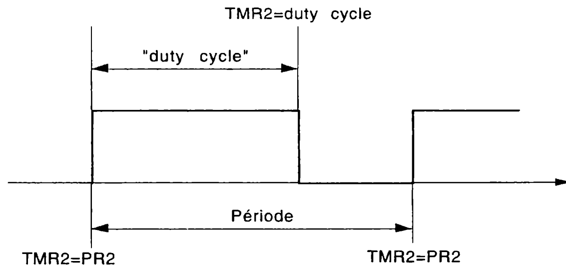


Figure 12.4. *L'impulsion modulée en largeur*

12.9.3.2. *Réalisation de la période*

La période est spécifiée en chargeant PR2.

Période = (PR2+1) x 4 x pré-division de TMR2

12.9.3.3. *Réalisation du « duty cycle »*

La valeur du « duty cycle » est spécifiée en chargeant le registre CCPRxL (huit bits de poids faibles) et les bits 4 et 5 du registre CCPxCON (deux bits de poids forts) afin d'obtenir une résolution sur 10 bits.

Valeur du « duty cycle » = (CCPRxL:CCPxCON(4:5)) x TOSC x pré-division de TMR2

Les valeurs de programmation du module peuvent être chargées à tout moment mais la donnée n'est pas transférée dans CCPRxH tant que l'égalité entre PR2 et TMR2 n'est pas apparue.

12.9.3.4. *Mode opératoire*

– écrire la valeur de la période dans le registre PR2 ;

– écrire la valeur du « duty-cycle » dans le registre CCPRxL et les bits 4 et 5 du registre CCPxCON ;

– mettre la ligne CCPx en sortie ;

– déterminer la pré-division de TMR2 et mettre en service le compteur TMR2 avec le bit T2CON ;

– configurer le module CCPx pour l'opération de modulation de largeur d'impulsion.

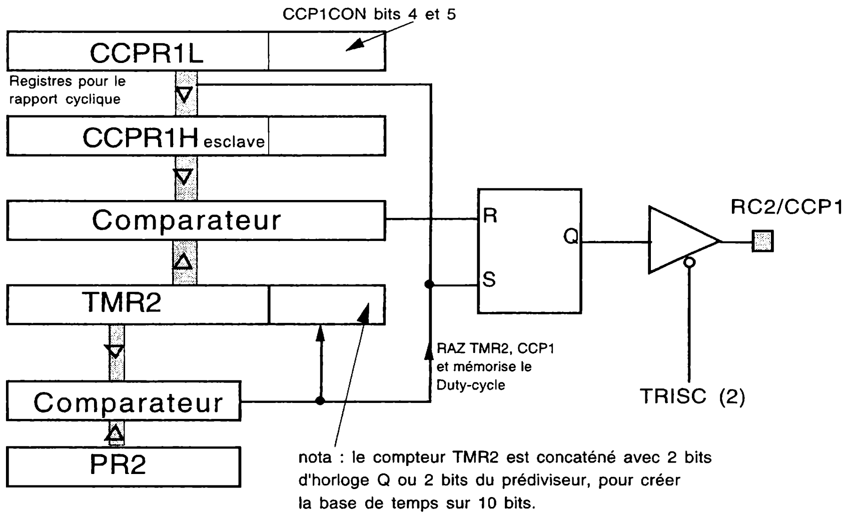


Figure 12.5. *Structure du générateur d'impulsions*

Chapitre 13
La liaison SPI du 16F877

SPI signifie *Serial Peripheral Interface.*

Généralités : cette liaison est une liaison série synchrone qui fait partie du module MSSP. Elle transmet à la fois les données (bits) mais aussi un signal d'horloge qui synchronise la réception de ces données. Cette liaison permet de relier le microcontrôleur à de nombreux périphériques intelligents équipés de registres à décalage tels que afficheur, EEPROM à liaison série, convertisseur, etc. Dans ce mode de liaison, utilisable en réseau, il y a un maître et un ou plusieurs esclaves, c'est le maître qui génère l'horloge nécessaire au transfert en lecture depuis l'esclave ou en écriture vers l'esclave.

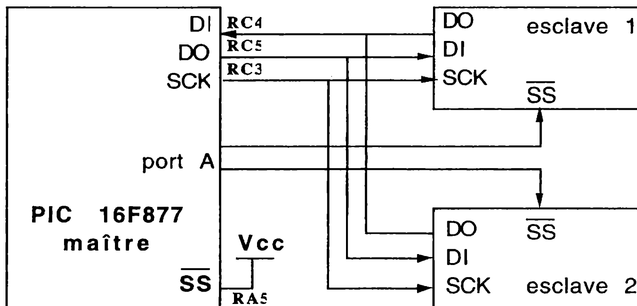


Figure 13.1. *Exemple de câblage d'une liaison SPI*

13.1. Constitution

Cette interface est pilotée par deux registres (voir figure 13.2) de la MSSP :

- le registre de contrôle SSPCON adresse 14h ;
- le registre d'état SSPSTAT adresse 94h.

Définition des connexions

SDO (RC5) : *Serial Data Out* – sortie de donnée ;

SDI (RC4/SDA) : *Serial Data Input* – entrée de donnée ;

SCK (RC3/SCL) : *Serial ClocK* – sortie d'horloge sur maître ;
– entrée d'horloge sur esclave.

SS (RA5/AN4) : *Slave Select*

Cette ligne de sélection, qui fait partie du port A, active au niveau bas, permet au maître d'activer la SPI de l'esclave avec qui dialoguer dans un réseau. En cas de dialogue entre plusieurs microcontrôleurs, si la connexion SS est au niveau haut le microcontrôleur est maître, et, à l'inverse, si SS est au niveau bas le microcontrôleur est esclave. Toutefois, s'il n'y a qu'un esclave, l'usage de la connexion SS n'est pas obligatoire.

13.2. Fonctionnement de l'interface

Grâce au câblage du registre à décalage on peut émettre et recevoir simultanément, quand une donnée est envoyée une autre est reçue. Le transfert est contrôlé par le maître, quand un maître veut recevoir une donnée d'un esclave, il lui faut écrire un mot dans son *buffer*, même sans signification, afin de générer l'horloge indispensable à l'esclave.

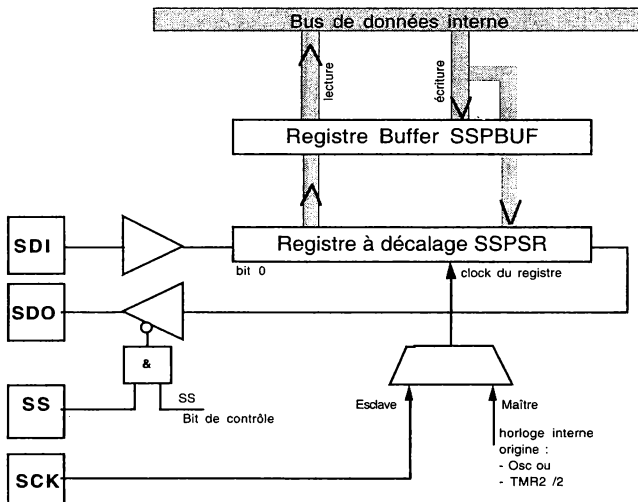


Figure 13.2. *Structure de la SPI*

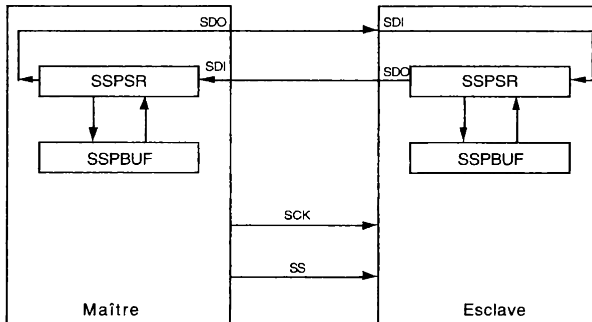


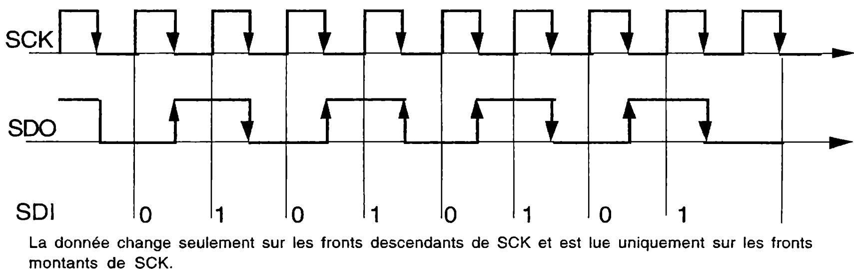
Figure 13.3. *Interfaçage entre maître et esclave*

Quand un mot est transmis, il faut lire le *buffer* avant de transmettre à nouveau même si la donnée reçue est sans intérêt.

Un bit de donnée change sur un front de l'horloge SCK (registre à décalage), et il est verrouillé en réception sur le front opposé de SCK.(voir figure 13.4.)

13.2.1. *Programmation des lignes de l'interface*

Les sens de transfert des connexions de l'interface sont à programmer ;



La donnée change seulement sur les fronts descendants de SCK et est lue uniquement sur les fronts montants de SCK.

Figure 13.4. *Chronogrammes de communication*

SDO (RC5) est une sortie il faut donc mettre le bit 5 de TRISC à 0.

Si SCK est une entrée pour l'esclave TRISC(3) doit être à 1.

Si SCK est une sortie pour le maître TRISC(3) doit être à 0.

La ligne SDI est contrôlée automatiquement par le module SPI.

Pour SS, la ligne RA5 du port A doit être configurée comme une entrée/sortie numérique dans le registre de contrôle ADCON1.

Si une connexion n'est pas utilisée, elle doit être programmée en sens inverse de sont utilisation normale.

13.2.1.1. *Le registre de contrôle SSPCON*

SSPOV : *Synchronous Serial Port OVerflow bit*

SSPCON		SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
14h		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

Pour l'interface SPI : ce bit passe à 1 quand un nouveau mot est reçu alors que le buffer SSPBUF contient la donnée précédente (débordement). La donnée dans le registre à décalage (SSPSR) est perdue. En mode esclave, le programme doit lire le SSPBUF afin d'éviter le débordement. En mode maître le bit n'est pas mis à 1 car le transfert est dû à l'écriture dans le SSPBUF. Ce bit doit être remis à 0 par programme.

SSPEN : *Synchronous Serial Port ENable bit*

A 1 il met en service le port série et configure les lignes SCK, SDO, SDI et SS selon les bits SSPM3 à SSPM0. Dans ce cas il faut que les lignes soient configurées correctement en entrées et en sorties.

CKP : *ClocK Polarity select bit*

Détermine l'état de repos de l'horloge.

A 1 l'état de repos est un niveau 1 et inversement.

SSPM3 à SSPM0 : *Synchronous Serial Port Mode Select bits*

M3	M2	M1	M0	Origine de l'horloge
0	0	0	0	Fosc/4
0	0	0	1	Fosc/16
0	0	1	0	Fosc/64
0	0	1	1	TMR2 output/2
0	1	0	0	entrée SCK - SS en service
0	1	0	1	entrée SCK - SS libre

Pour la SPI en maître ces bits permettent de déterminer l'origine (TMR2 ou OSC) et la fréquence de l'horloge de la transmission, pour la SPI en esclave ils mettent en service ou non la ligne SS.

13.2.1.2. *Le registre d'état : SSPSTAT*–Synchronous Serial Port STATus register

SSPSTAT	SMP	CKE						BF
94h	R/W-0	R/W-0						R-0

Le registre SSPSTAT permet de suivre le bon fonctionnement de la transmission.

SMP : *data SaMPle timing*
Echantillonnage d'un bit sur l'entrée SDI.
– SPI mode maître
SMP = 1 : l'échantillonnage de la donnée reçue a lieu à la fin du bit de donnée.
SMP = 0 : l'échantillonnage a lieu au milieu du bit.
Ce réglage de l'échantillonnage permet d'être compatible avec tout esclave.
– SPI mode esclave
Dans ce mode de fonctionnement SMP doit être mis à 0.

CKE : *SPI ClocK Edge select*
Si CKP = 0 : repos de l'horloge au niveau 0
1= la donnée est transmise sur un front montant de l'horloge SCK.
0 = la donnée est transmise sur un front descendant de SCK.
Si CKP = 1 : repos de l'horloge au niveau 1
1 = la donnée est transmise sur un front descendant de SCK
0 = la donnée est transmise sur un front montant de SCK.

BF : *Buffer Full Status bit*
Indique quand il est à 1 que le registre de réception (SSPBUF) est plein. Il faut lire le buffer avant toute autre lecture ou écriture aussi bien pour le maître que pour l'esclave.

13.2.1.3. *Le registre PIR1*

Seul le bit SSPIF du registre PIR1 est utilisé ; il indique la fin d'un transfert dans l'interface en passant à 1. Une interruption peut être demandée si SSPIE du registre PIE1 l'autorise.

13.3. Utilisation de l'interface

Pour mettre en service l'interface SPI, il faut mettre à 1 le bit SSPEN (MSSP *Enable bit*) du registre SSPCON.

Durant un transfert, des données sont simultanément transmises et reçues.

La structure de l'interface comprend deux registres, un registre à décalage SSPBSR et un tampon de lecture SSPBUF. Le registre SSPSR est utilisé pour émettre et recevoir. En réception, les deux registres sont utilisés alors qu'en transmission seul le registre à décalage connecté directement au bus interne est utilisé. Quand le mot est reçu, il est transféré dans le buffer SSPBUF s'il est correct, les anomalies du mot reçu sont signalées dans le registre d'état.

PIR1	-	-	-	-	SSPIF	-	-	-
0Ch	-	-	-	-	R/W-0	-	-	-

PIE1	-	-	-	-	SSPIE	-	-	-
8Ch	-	-	-	-	R/W-0	-	-	-

Une écriture dans le registre suffit pour émettre, le bit SSPIF passe à 1 à la fin de l'émission et doit être remis à 0 avant d'émettre à nouveau. Une nouvelle donnée ne peut donc être envoyée si le précédent transfert n'est pas terminé (écrasement).

Une horloge générée par le maître synchronise le décalage et l'échantillonnage sur les deux lignes de données (SDI et SDO).

13.3.1. *Contrôle de phase d'horloge et de polarité*

L'horloge peut être au repos à l'état haut ou à l'état bas selon les caractéristiques du composant avec qui le dialogue s'effectue, ceci est déterminé par le bit CKP du registre de contrôle SSPCON. Dans un transfert c'est le bit de poids fort MSB qui est transmis en premier.

Le choix du mode de transfert dépend du bit CKE du registre de contrôle.

13.3.1.1. *Fonctionnement en mode esclave (voir figure 13.5.)*

La configuration de l'interface détermine les caractéristiques du transfert.

L'horloge générée par le maître arrive par la connexion SCK.

Si CKE = 1

Le transfert commence quand SS passe à 0 (la ligne SS est active) et s'arrête quand SS repasse à 1. Le premier bit de la donnée (MSB) est échantillonné sur le premier front actif qui suit le passage à 0 de la ligne SS. Le bit SSPIF de PIR1 est mis à 1 quand le dernier bit de la donnée est verrouillé. La ligne SS doit donc être remise à 1 entre chaque octet transféré.

Si CKE = 0

Le transfert commence avec la première impusion d'horloge SCK, c'est-à-dire avec le front actif du premier cycle de SCK. Le transfert est terminé quand le bit SSPIF passe à 1. Dans ce cas la ligne SS peut rester à 0 pendant le transfert de plusieurs octets.

Quand le microcontrôleur esclave est en sommeil, il est réveillé quand un mot est reçu.

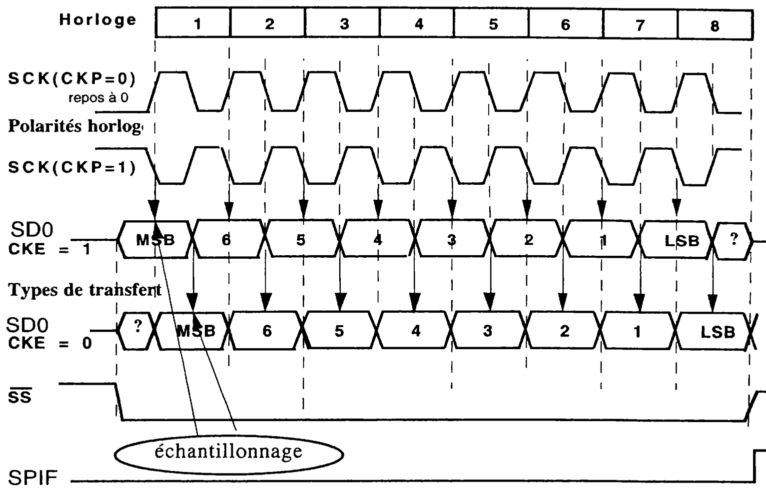


Figure 13.5. *Chronogrammes de transfert dans l'esclave (SMP = 0)*

13.3.1.2. *Le mode maître*

Le maître peut émettre à tout moment car il contrôle l'horloge. La fréquence de l'horloge dépend des bits SSPM0 à SSPM3 du registre SSPCON.

Pour le maître, le transfert commence quand une donnée est écrite dans le SSPBUF et se termine quand le bit SSPIF passe à 1 dans le registre PIR1.

Si l'esclave ne peut que recevoir, on peut programmer la ligne SDI du maître en entrée car elle est inutile.

En mode maître, l'échantillonnage continue sur l'entrée SDI pendant l'émission, le microcontrôleur peut donc recevoir des données par l'entrée SDI et les charger dans le *buffer*. L'échantillonnage sur la ligne SDI dépend du bit SMP du registre SSPSTAT ; si SMP = 0 l'échantillonnage a lieu au milieu du cycle d'horloge SCK et si SMP = 1 l'échantillonnage a lieu à la fin de SCK (voir figure 13.5).

13.4. Les erreurs durant un transfert

Une erreur de collision peut se produire quand on essaie d'écrire dans le registre de données SSPBUF alors qu'un transfert est en cours, dans ce cas le bit WCOL du registre SSPCON passe à 1. La donnée présente dans le registre à décalage est perdue.

Le programme de gestion doit donc lire la donnée dans le registre SSPBUF.

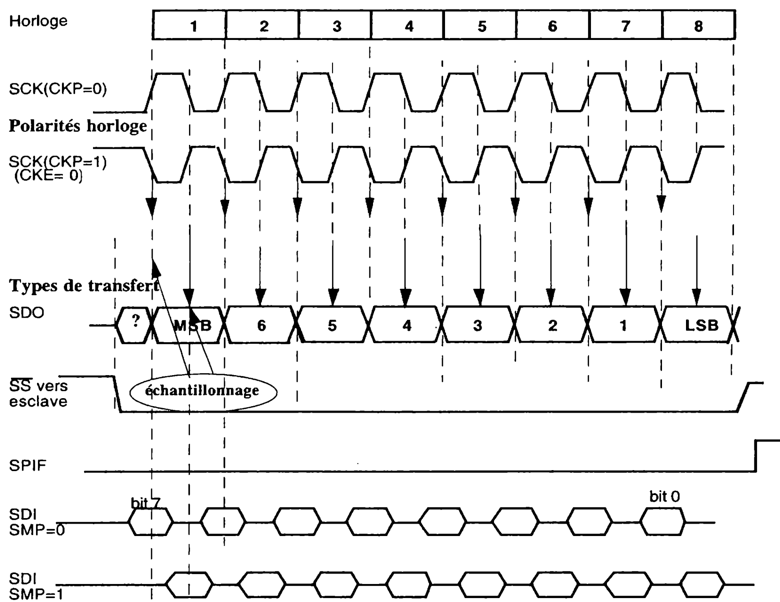


Figure 13.6. *Chronogrammes du maître*

13.5. Les interruptions

PIR1	PSPIF	ADIF	RCIF	TXIF	**SSPIF**	CCPI1IF	TMR2IF	TMR1IF
0Ch	R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0

Le passage à 1 du bit SSPIF du registre PIR1 qui indique la fin d'un transfert, peut provoquer une demande d'interruption. Celle-ci est prise en compte si elle est autorisée par le bit SSPIE du registre PIE1, par le masque d'interruption des périphériques PEIE du registre INTCON et par le bit GIE (masque global).

PIE1	-	-	-	-	**SSPIE**	-	-	-
8Ch	-	-	-	-	0	-	-	-

INTCON	GIE	PEIE	-	-	-	-	-	-
0Bh	R/W-0	R/W-0	-	-	-	-	-	-

REMARQUE.– Le microcontrôleur 16F877 possèdant 4 blocs de mémoire de travail, il est nécessaire d'utiliser systématiquement PCLATH, en conséquence dans le sous programme d'interruption il faut sauvegarder l'accumulateur W, le registre d'état STATUS et le registre PCLATH.

13.6. Exemple d'utilisation

Un microcontrôleur 16F877 envoie à un autre PIC un octet, cet octet reçu est affiché sur les huit leds connectées au port B.

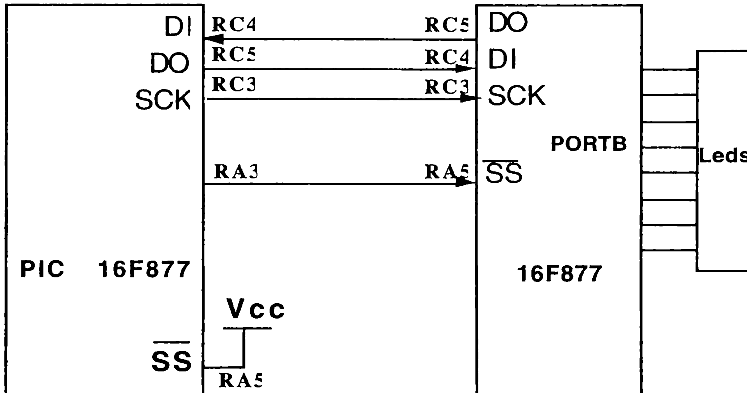


Figure 13.7. *Transmission SPI entre deux microcontrôleurs*

13.6.1. *Programme du maître*

13.6.1.1. *Etude des registres*

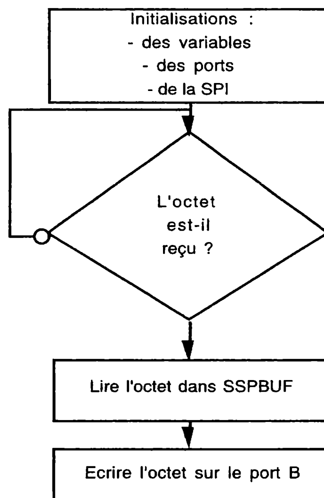


Figure 13.8. *Organigramme d'envoi*

Toutes les lignes du port A sont en sorties, TRISA = 0

La ligne SCK (RC3) est en sortie, la ligne SDI (RC4) en entrée et la ligne SDO (RC5) en sortie donc TRISC = 10h.

Le mode convertisseur du port A est désélectionné donc ADCON1 = 06h.

La SPI est en service SSPEN est à 1, l'horloge est au repos CKP est à 0 et l'horloge interne est divisée par 16 donc SSPCON = 31h.

SMP = 0 ; l'échantillonnage se fait au milieu de la période d'horloge et CKE = 1 et l'horloge est active sur front descendant donc SSPSTAT= 40h.

13.6.1.2. *Source du logiciel maître*

```
; Liaison SPI connexion entre 2 PICs
; Le maître envoie un octet à l'esclave
    list p=16F877
    #include "p16F877.inc"

    __CONFIG _CP_OFF & _DEBUG_OFF & _WRT_ENABLE_OFF & _CPD_OFF &
_LVP_OFF & _BODEN_OFF & _PWRTE_ON & _WDT_OFF & _XT_OSC


#define SS PORTA,2          ; sélection esclave par RA2
octet equ 55h               ; octet à transmettre

    ORG 0
    bcf STATUS,RP1
; Initialisation des ports
    bsf STATUS,RP0          ; bloc 1
    movlw 0x00              ; PORTA en sorties
    movwf TRISA             ;
    movlw 0x06              ; désactivation de A/D
    movwf ADCON1            ; sur PORTA
; Configuration de la SPI
    bsf STATUS,RP0          ; bloc 1
    movlw B'00010000'       ; SCK en sortie (Master), SDI en entrée
    movwf TRISC             ; SDO en sortie
    movlw B'01000000'       ; échantillonnage
    movwf SSPSTAT           ; au milieu et front montant
    bcf STATUS,RP0          ; bloc 0
    movlw B'00110001'       ; repos à 1 , 1/16 Tosc
    movwf SSPCON            ; et SSP en service
; mise en service SPI esclave
    bcf SS                  ; activation de l'esclave
; Envoi de la donnée
    movlw octet             ; octet dans W
    movwf SSPBUF            ; écriture dans SSPBUF
    bsf STATUS,RP0          ; bloc 1
```

```
test btfss SSPSTAT,BF    ; Transfert terminé? (BF?)
     goto test           ; teste encore
     bcf STATUS,RP0      ; bloc 0
     movf SSPBUF,W       ; lecture du  SSPBUF, donnée non utile
     bsf SS              ; déselection de l'esclave
Fin  goto Fin
     END
```

13.6.2. *Programme de l'esclave*

13.6.2.1. *Etude des registres*

Toutes les lignes du port B sont en sorties donc TRISB = 0.
La ligne RA5 est en entrée donc TRISA = 10h.
Le convertisseur est désélectionné donc ADCON1 = 06h.

La SPI est en service, l'horloge est au repos à 1 et la ligne SS est en service donc SSPCON = 34h.

L'échantillonnage a lieu au milieu de la période d'horloge et le front descendant de l'horloge est actif donc : SSPSTAT = 40h, CHE=1 et SMP=0.

```
; Le maitre envoi un octet, l'esclave reçoit l'octet et l'affiche
```

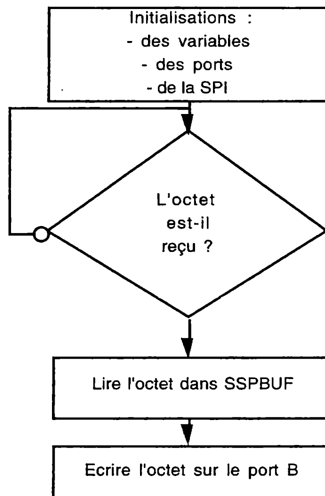


Figure 13.9. *Réception*

```
sur le port B

     list p=16F877
     #include "p16F877.inc"
```

```
    __CONFIG _CP_OFF & _DEBUG_OFF & _WRT_ENABLE_OFF & _CPD_OFF &
_LVP_OFF & _BODEN_OFF & _PWRTE_ON & _WDT_OFF & _XT_OSC
    ORG 0
    bcf STATUS,RP1      ;
; Initialisation de la  SPI
    bsf STATUS,RP0      ; bloc 1
    movlw D'5'          ; SS en entrée
    movwf TRISA         ;
    movlw 0x06          ; désélection du convertisseur CAN
    movwf ADCON1        ;
; Configuration du portB
    movlw 0x00          ; Configuration PORTB
    movwf TRISB         ; en sorties
; Configuration de la SPI
    bsf STATUS,RP0      ; bloc 1
    movlw B'00011000'   ; SCK entrée (Slave), SDI entrée
    movwf TRISC         ; SDO en sortie
    movlw B'01000000'   ; échantillonnage bit
    movwf SSPSTAT       ; au milieu
    bcf STATUS,RP0      ; bloc 0
    movlw B'00110100'   ; Mode 1,1 SPI esclave, SS en service
    movwf SSPCON        ; interface en service
; Attente d'un caractère
Test movlw SSPSTAT      ; Adressage indirect
    movwf FSR           ; test dans SSPSTAT
    btfss INDF,BF       ; teste BF
    goto Test           ; Nouveau test
    bcf STATUS,RP0      ; bloc 0
    movf SSPBUF,W       ; mettre SSPBUF dans W
    bcf STATUS,RP0      ; bloc 0
    movwf PORTB         ; Octet sur PORTB
    END
```

13.7. Interfaçage d'un convertisseur analogique numérique MAX 187

Le câblage et les chronogrammes de transfert sont représentés ci-dessous.

La conversion donne une valeur sur 10 bits donc deux octets. Pour lancer la conversion il faut faire passer la liaison CS à 0, puis attendre qu'elle soit terminée, ce qui est signalé par la remontée à 1 de DOUT.

Solution

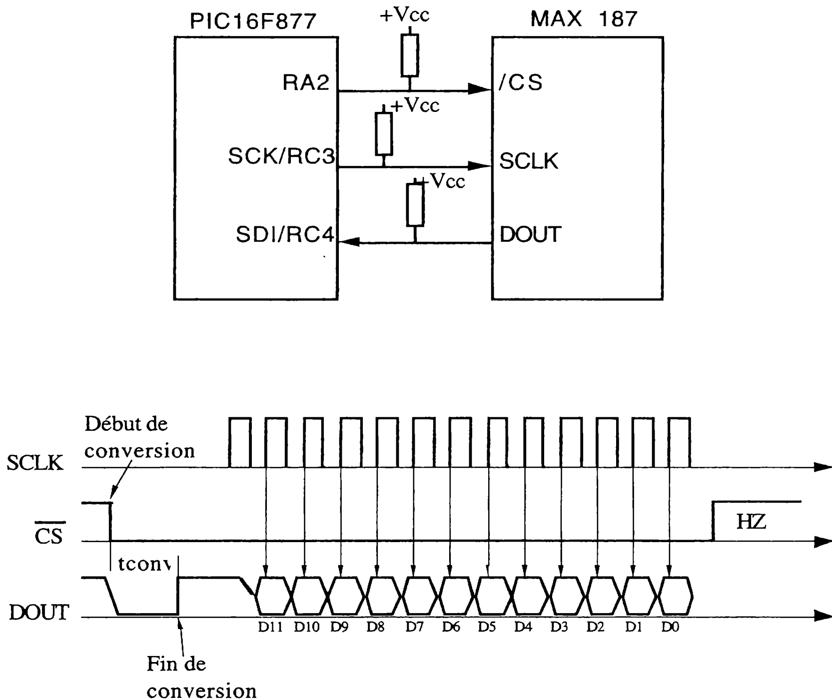


Figure 13.10. *Liaison avec le convertisseur MAX 187*

Le niveau de repos de l'horloge est 0, la fréquence d'horloge sera réglée sur 1/16 de la fréquence du microcontrôleur et on réalisera l'échantillonnage des données au milieu. Pour générer les signaux d'horloge indispensables à l'esclave (MAX 187) on écrira des octets fictifs dans le buffer. Les données issues de la conversion seront affichées sur les ports B et C afin de contrôler le bon fonctionnement.

Source d'une solution :

```
    list P=16F877
    #include "P16F877.inc"

    __CONFIG _CP_OFF & _DEBUG_OFF & _WRT_ENABLE_OFF & _CPD_OFF &
_LVP_OFF & _BODEN_OFF & _PWRTE_ON & _WDT_OFF & _XT_OSC


#define CS PORTA,2          ; sélection esclave par RA2
octet equ 0xFF              ; octet fictif à transmettre pour horloge
```

```
    CBLOCK 0x0C
    octet1 : 1          ; réservations pour résultats
    octet2 : 1          ; 2 octets
    ENDC

    ORG 0
    bcf STATUS,RP1
; Initialisation des ports
    bsf STATUS,RP0      ; bloc 1
    movlw 0x00          ; PORTA en sorties
    movwf TRISA         ; PORTB en sorties
    movwf TRISB
    movwf TRISC         ; PORTC en sorties
    movlw 0x06          ; désactivation de A/D
    movwf ADCON1        ; sur PORTA
; Configuration de la SPI
    bsf STATUS,RP0      ; bloc 1
    movlw B'00010000'   ; SCK en sortie (Master), SDI en entrée
    movwf TRISC         ; SDO en sortie
    movlw B'01000000'   ; échantillonnage
    movwf SSPSTAT       ; au milieu et front montant
    bcf STATUS,RP0      ; bloc 0
    movlw B'00110001'   ; repos à 0 , 1/16 Tosc
    movwf SSPCON        ; et SSP en service
; mise en service SPI esclave
    bcf CS              ; mise en marche conversion
; attente fin de conversion test sur SDI
finconv  btfss PORTC,4  ; lecture du portc
    goto finconv        ; si SDI=0 recommence
; lecture 1er octet
    movlw octet         ; octet fictif dans W
    movwf SSPBUF        ; écriture dans SSPBUF pour horloge
    bsf STATUS,RP0      ; bloc 1
test btfss SSPSTAT,BF   ; Octet reçu ? (BF=1?)
    goto test           ; teste encore
    bcf STATUS,RP0      ; bloc 0
    movf SSPBUF,W       ; lecture du SSPBUF 1er octet
    movwf octet1        ; mise en mémoire
; lecture du 2eme octet
    movlw octet         ; octet fictif dans W
    movwf SSPBUF        ; écriture dans SSPBUF pour l'horloge
    bsf STATUS,RP0      ; bloc 1
test2 btfss SSPSTAT,BF  ; Transfert terminé? (BF=1?)
    goto test2          ; teste encore
    bcf STATUS,RP0      ; bloc 0
```

```
    movf SSPBUF,W         ; lecture du SSPBUF 2ème octet
    movwf octet2
    bsf CS                ; déselection de l'esclave
    movf octet1,w
    movwf PORTB           ; affichage sur le port B poids faible
    movf octet2,w
    andlw 0x0f            ; élimine les bits non significatifs
    movwf PORTC           ; affichage sur port C
Fin goto Fin
    END
```

Chapitre 14
La liaison I2C

14.1. Généralités

14.1.1. *Rappel*

La liaison I2C est une liaison synchrone à 2 fils qui peut être utilisée en réseau avec un ou plusieurs maîtres et de nombreux esclaves et dont les connexions sont de type « drain ouvert ». Au repos les deux lignes rappelées à +VCC par des résistances sont à l'état haut ; un début de transmission est déclaré par une séquence de START (le maître prend le contrôle de la ligne), puis le maître envoi un octet d'adresse, comprenant un bit de lecture ou d'écriture, sur le bus. L'esclave qui a reconnu son adresse répond alors par un accusé de réception (ACK). Ensuite la transmission s'effectue dans le sens fixé par le bit R/W situé dans l'adresse. Après chaque octet transmis, le récepteur qu'il soit maître ou esclave envoi un accusé de réception à l'émetteur. Quand une transmission est terminée, la liaison est remise à l'état de repos par le maître en générant une séquence de STOP. Le maître libère le bus.

C'est l'horloge (SCL) générée par le maître qui synchronise les échanges quel que soit le sens de la transmission. Le bit de poids fort de la donnée est transmis en premier. Un bit de donnée n'est considéré comme valide que pendant le niveau haut du bit d'horloge, le changement d'état se fait pendant le niveau bas de l'horloge.

14.1.2. *La liaison I2C du PIC 16F877*

Cette liaison synchrone fait partie du MSSP (*Master Synchronous asynchronous Serial Port*), elle permet les connexions en réseau avec plusieurs maîtres et des possibilités d'interruption sur les bits de start et de stop. Cette liaison possède les spécifications du mode standart avec adressage des esclaves sur 7 ou 10 bits, et un fonctionnement de 100 KHz jusqu'à 400 KHz.

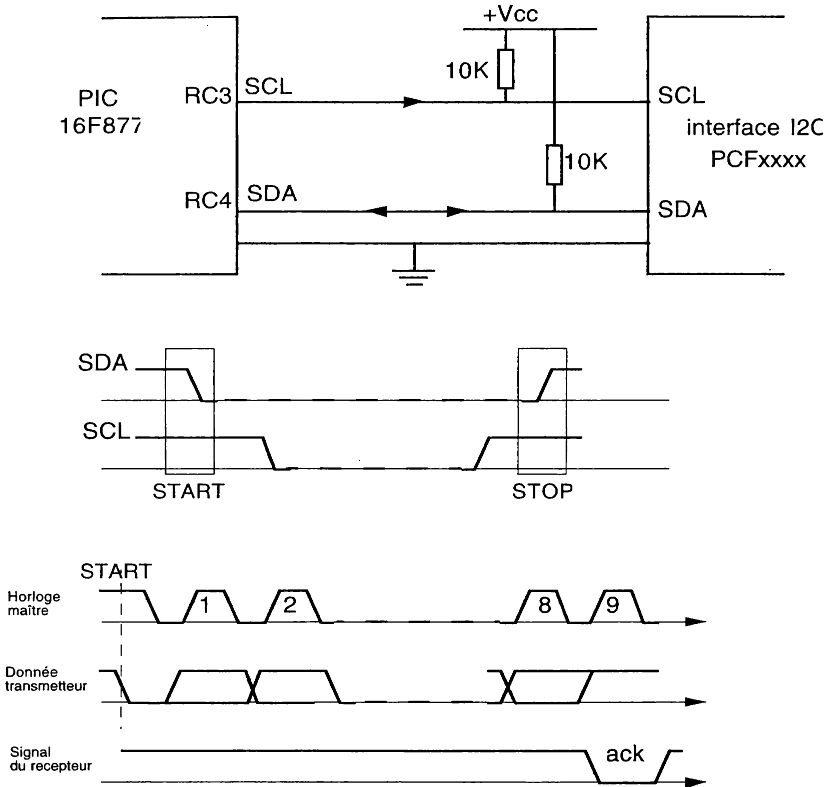


Figure 14.1. *Bus I2C (généralités)*

Les lignes SDA (donnée) et SCL (horloge) respectivement RC4 et RC3 du PIC 16F877 sont rappelées au potentiel de l'alimentation par des résistances.

Ces connexions sont automatiquement mises en service par la mise à 1 du bit SSPEN du registre SSPCON. Toutefois l'utilisateur doit configurer en entrées les lignes avec les bits correspondants du registre TRISC.

Dix registres sont utilisés pour cette liaison :

- SSPCON : registre de contrôle (14h) ;
- SSPCON2 : registre de contrôle 2 (91h) ;
- SSPSTAT : registre d'état de la liaison série (94h) ;
- SSPBUF : buffer de lecture écriture (13h) ;
- SSPSR : registre à décalage non accessible ;

– SSPADD : registre d'adresses ou générateur de bauds (93h) ;

– PIR1 (0Ch) pour le bit SSPIF, PIE1 (8Ch) pour le bit SSPIE, PIR2 (0Dh) pour le bit BCLIF, et PIE2 (8Dh)pour le bit BCLIE.

14.2. Etude des registres

14.2.1. *Le registre SSPCON*

SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
14h	R/W-0	R/W-0	R/W-0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0

R : lecture, W : écriture, x : inconnu, u : inchangé, 1 : mis à 1 au reset, 0 : mis à 0 au reset

Les bits SSPM3 à SSPM0 déterminent les modes de fonctionnement de cette liaison.

M3	M2	M1	M0	
0	1	1	0	I2C en mode esclave avec adressage sur 7 bits
0	1	1	1	I2C en mode esclave avec adressage sur 10 bits
1	0	0	0	I2C en mode maître avec une fréquence d'horloge = Fosc/(4*(SSPADD+1))
1	0	1	1	I2C mode maître contrôlé par logique interne (esclave en attente)
1	1	1	0	I2C mode maître contrôlé par logique interne, adresse sur 7 bits et inter ruptions autorisées sur bit de START et de STOP.
1	1	1	1	I2C mode maître contrôlé par logique interne, adresse sur 10 bits et inter ruptions autorisées sur bit de START et de STOP.

WCOL : indicateur de collision.

En mode maître, ce bit passe à 1 si une tentative d'écriture dans le buffer (SSPBUF) a lieu quand les conditions requises pour l'I2C ne sont pas valides.

En mode esclave, ce bit passe à 1 quand une écriture dans le buffer a lieu durant la transmission du mot précédent. Ce bit doit être remis à 0 par programme.

SSPOV : indicateur de dépassement.

Passe à 1 quand un octet est reçu alors que le buffer contient la donnée précédente. Ce bit n'est pas utile en transmission et doit être effacé par logiciel.

SSPEN : mise en service du port série SSP.

Sa mise à 1 provoque la configuration des connexions du port I2C selon les bits SSPM3 à SSPM0.

CKP : polarité de l'horloge.

En mode esclave, CKP à 1 met simplement la ligne d'horloge SCL en service et CKP à 0 maintient la ligne SCL au niveau bas.

En mode maître ce bit n'est pas utilisé.

14.2.2. *Le registre SSPCON2 : registre de contrôle du port série synchrone*

SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
91h	R/W-0	R/W-0	R/W-0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0

R : lecture, W : écriture, x : inconnu, u : inchangé, 1 : mis à 1 au reset, 0 : mis à 0 au reset

GCEN : *General Call ENable bit*

1 : autorise une interruption quand une adresse générale (0000h) a été reçue dans le registre à décalage (mode I2C uniquement).

0 : l'adresse générale n'est pas en service.

ACKSTAT : *ACKnoledge STATus bit*

Pour le maître en émission, il indique la valeur de l'accusé de réception reçu. Normalement il doit être à 0 sinon cela signifie que le récepteur n'a pas envoyé d'accusé de réception.

ACKDT : *ACK DaTa bit*

Ce bit contient la valeur qui est transmise à l'esclave quand l'utilisateur lance une séquence d'accusé de réception à la fin d'une réception par le maître. Ce bit doit être à 0 pour un accusé de réception valide.

ACKEN : *ACKnoledge sequence ENable*

Pour le maître en réception :

Mis à 1 : génération d'une séquence d'accusé de réception par le maître.

Ce bit est automatiquement remis à 0 par le microcontrôleur.

RCEN : *ReCeive ENable bit*

Mis à 1 : le maître est mis en réception.

PEN : *STOP condition ENable*

Mis à 1 : le maître génére une séquence de STOP sur les lignes SDA et SCK, et ce bit est remis à 0 automatiquement par le composant.

RSEN : Repeated Start condition ENable

Mis à 1 : le maître (re)-génère un signal START sur SDA et SCK quelque soit l'état de la ligne SDA. Ce bit est remis à 0 automatiquement par le composant.

SEN : START condition ENable bit.

Mis à 1 : le maître génère un signal de START sur SDA et SCK et ce bit est remis à 0 automatiquement par le composant.

Nota : si le module I2C n'est pas en attente, les bits ACKEBN, RCEN, PEN, RSEN et SEN ne peuvent être mis à 1 et le SSPBUF ne peut être chargé.

14.2.3. *Le registre SSPSTAT : registre d'état de la liaison série synchrone*

SSPSTAT	SMP	CKE	D/A	P	S	R/W	UA	BF
94h	R/W-0	R/W-0	R/W-0	R/W -0	R/W -0	R/W -0	R/W -0	R/W -0

R : lecture, W : écriture, x : inconnu, u : inchangé, 1 : mis à 1 au reset, 0 : mis à 0 au reset

SMP : *SaMple bit*

1 : mode standart à 100 KHz, le contrôle de slew rate n'est pas activé.

0 : mode grande vitesse à 400 KHz, le contrôle de slew rate est activé.

CKE : pour la liaison I2C

1 : les niveaux d'entrée sont conformes au bus SMBus.

0 : les niveaux d'entrée sont conformes au bus I2C.

D/A : *Data/Address bit*

1 : indique que le dernier mot reçu ou émis est une donnée.

0 : indique que le dernier mot reçu ou émis est une adresse.

P : bit de STOP

1 : indique qu'un bit de stop a été détecté.

0 : aucun bit de stop détecté.

S : bit de START

1 : indique qu'un bit de start a été reconnu.

0 : pas de bit de START détecté.

R/W : information de lecture ou d'écriture

Ce bit contient la valeur du bit R/W de la dernière adresse reçue.

En mode esclave : 1 => lecture
0 => écriture

En mode maître : 1 => transmission en cours
0 => pas de transmission

UA : *Update address* en mode d'adressage 10 bits uniquement

Indique, quand il est à 1, que l'utilisateur doit mettre l'octet de poids faible de l'adresse dans le registre d'adresse SSPADD.

BF : *Buffer SSPBUF* plein

En réception : à 1 il indique que la réception d'un octet est terminée le buffer SSPBUF est plein.

En transmission : à 1 il indique que la transmission est en cours ; le buffer SSPBUF est plein (sans tenir compte de ACK ni de STOP), à 0 le buffer est vide donc la transmission est terminée.

14.2.4. *Le registre SSPBUF (13h)*

C'est par ce registre que les données sont émises ou reçues *via* le bus de données.

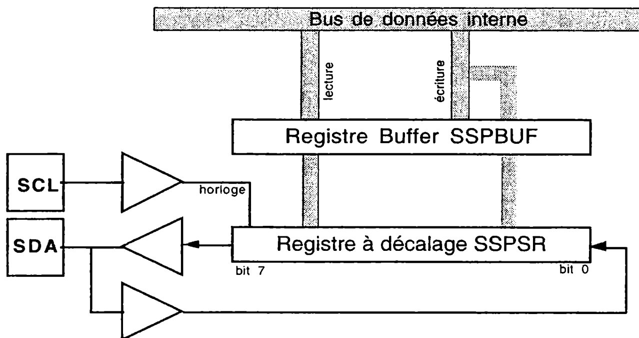


Figure 14.2. *Structure du module I2C*

14.2.5. *Le registre SSPSR*

SSPSR est le registre à décalage pour le transfert en série.

En réception, SSPBUF et SSPSR forment un double buffer en réception, ce qui permet la réception de la donnée suivante avant la lecture du dernier mot reçu.

Quand le mot complet est receptionné dans le SSPSR, il est transféré dans le SSPBUF et le bit SSPIF est mis à 1.

Si un autre mot est reçu avant que le buffer ne soit lu, il y a débordement, le bit SSPOV du registre SSPCON passe à 1 et le mot dans SSPSR est perdu.

14.2.6. *Le registre SSPADD (93h)*

En mode esclave ce registre contient l'adresse de l'esclave afin de la comparer à celle qui est transmise par le maître.

Si l'adresse est en mode 10 bits, l'utilisateur doit d'abord écrire le mot de poids fort « 1111 - 0 - A9-A8- 0 » et ensuite après la reconnaissane de l'octet de poids fort, le mot de poids faible « A7 - A0 » doit être chargé.

En mode maître, il permet de régler la durée du générateur de bauds. Pour cela il faut le charger avec une valeur binaire (SSPADD) résultat du calcul :

Baud Rate = Fosc / (4*(SSPADD + 1))

Fosc est la fréquence du quartz et Baud Rate correspond à la vitesse de transmission qui dépend des composants du réseau.

Le générateur de Bauds est un décompteur qui passe de la valeur de SSPADD à 0 et détermine le temps TBRG.

14.2.7. *Les registres PIR1 (0Ch) et PIE1 (8Ch)*

Le bit 3 (SSPIF) du registre PIR1 à l'adresse 0Ch est un indicateur de fin de tâche associé au fonctionnement du module MSSP en mode SPI ou en mode I2C. Ce bit doit être remis à 0 avant le retour d'interruption.

Pour le mode I2C :

En esclave, ce bit à 1 indique qu'une transmission a été réalisée.

En maître il indique :

- qu'un bit de START a été généré ;
- qu'un bit de STOP a été généré ;
- qu'un RESTART a été généré ;
- qu'un ACK a été généré ;
- qu'un STOP ou un START a été détecté quand le module est en attente en mode multimaître.

Ce bit peut être utilisé pour connaître l'état de la liaison I2C afin d'enchaîner les commandes. Ce bit doit être remis à 0 par le programme.

Une interruption provoquée par SSPIF n'est prise en compte que si le bit SSPIE du registre PIE1 au niveau 1 l'autorise.

14.2.8. *Les registres PIR2 (0Dh) et PIE2 (8Dh)*

Le bit BCLIF est un indicateur de collision dans un réseau multimaître.

Une collision a lieu quand deux maîtres veulent prendre le contrôle du bus en même temps, il y a alors corruption des données. Chaque circuit vérifie en permanence l'état des lignes SDA et SCL, donc un maître détecte une collision si il ne relit pas l'état de la ligne qu'il cherche à imposer.

Une collision est détectée quand par exemple la ligne SDA est échantillonnée à 0 alors qu'elle devrait être à 1, dans ce cas le maître fait passer BCLIF du registre PIR2 à 1 et libère la liaison I2C. Cette collision peut avoir lieu durant un signal START, un START répété ou un STOP.

Le passage à 1 de ce bit BCLIF peut déclencher une interruption si le bit BCLIE du registre PIE2 au niveau 1 l'autorise.

14.3. Le fonctionnement en esclave

Les lignes SDA et SCL sont configurées en entrées. Quand une adresse est reconnue par comparaison ou quand une donnée est reçue le composant génère automatiquement l'impulsion d'accusé de réception ACK vers le maître et charge le buffer (SSPBUF) avec le contenu du registre à décalage (SSPSR).

ACK n'est pas généré si le bit BF (buffer plein) était à 1 avant le transfert ou si le bit SSPOV (débordement) était à 1.

Le bit BF est remis à 0 par la lecture du buffer SSPBUF et SSPOV doit être remis à 0 par logiciel.

14.3.1. *L'adressage sur 7 bits*

– le module en service attend le bit de START ;

– les 8 bits sont chargés dans le registre à décalage, les bits sont échantillonnés sur le front montant de l'horloge SCL ;

– la valeur du registre à décalage (SSPSR) est comparé au contenu du registre d'adresse (SSPADD), cette comparaison a lieu sur le front descendant de la 8e impulsion d'horloge ;

– si l'adresse est reconnue et si les bits BF et SSPOV sont à 0

- SSPSR est transféré dans SSPBUF sur le front descendant de la 8e impulsion d'horloge ;
- BF passe à 1 sur la 8e impulsion d'horloge ;

- ACK est généré ;

- le bit indicateur SSPIF du registre PIR1 passe à 1 et éventuellement une interruption est demandée, si elle est autorisée par le bit SSPIE, sur le front descendant de la 9e impulsion d'horloge.

14.3.2. *L'adressage sur 10 bits*

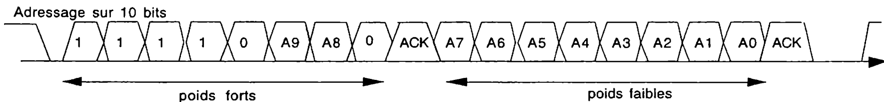


Figure 14.3. *Chronogrammes d'une adresse sur 10 bits*

Deux octets sont nécessaires à l'esclave.

Le premier mot est : 1111 0 A9 A8 0. Ce premier mot spécifie grâce à ses 5 bits de poids fort que l'adresse est sur 10 bits. A sa réception le bit R/W de SSPSTAT à 0 indique alors que l'esclave doit recevoir le 2e mot de l'adresse.

La séquence est dans ce cas :

– recevoir l'octet de poids fort (SSPIF, BF et UA passent à 1) ;

– mettre à jour SSPADD avec l'octet de poids faible en vue de la comparaison, ce qui met à 0 le bit UA et relâche la ligne d'horloge SCL ;

– lire le buffer pour mettre à 0 le bit BF puis remettre à 0 le bit SSPIF ;

– recevoir le 2e octet (poids faible) de l'adresse (SSPIF, BF et UA sont à 1) ;

– mettre à jour SSPADD avec l'octet de poids fort de l'adresse, ce qui met à 0 le bit UA et relâche la ligne d'horloge SCL ;

– lire le buffer SSPBUF pour remettre à 0 le bit BF puis remettre à 0 le bit SSPIF.

14.3.3. *L'adresse d'appel général*

Il existe une exception : c'est l'adresse générale constituée de tous les bits à 0 y compris R/W. Après la séquence de START, les huit bits sont décalés dans le registre à décalage et l'adresse est comparée au contenu du registre SSPADD. Dans ce cas tous les esclaves sont concernés, cette adresse est reconnue si le bit GCEN du registre SSPCON2 est à 1, dans ce cas SSPSR est tranféré dans SSPBUF, BF et SSPIF passent à 1. Après une telle adresse, l'octet suivant définit un certain nombre de fonctions particulières qui sont prises en compte par tous les récepteurs.

En mode 10 bits, SSPAD doit être mis à jour pour la 2ème moitié de l'adresse à reconnaître ; le bit UA de SSPSTAT est mis à 1. Si l'adresse générale est reçue quand GCEN est à 1 et alors que l'esclave est en mode 10 bits, la deuxième moitié de

l'adresse n'est évidemment pas nécessaire, UA n'est pas mis à 1 et l'esclave peut commencer à recevoir la donnée après l'accusé de réception ACK.

14.3.4. *Réception par l'esclave*

Si le bit R/W de l'adresse reçue est à 0 et l'adresse reconnue :

– le bit R/W de SSPSTAT est mis à 0 ;

– l'adresse reçue est transférée dans SSPBUF ;

– l'ACK est généré sauf si BF = 1 et SSPOV = 1 ;

– une interruption peut être demandée par le passage à 1 de SSPIF ;

– le bit SSPIF doit être remis à 0 par le programme.

REMARQUE.– Le module I2C peut recevoir et réveiller le microcontrôleur quand il reçoit une adresse reconnue ou une donnée complète si l'interruption est autorisée.

14.3.5. *Transmission par l'esclave*

Si le bit R/W de l'adresse reçue est à 1 et l'adresse reconnue :

– le bit R/W de SSPSTAT est mis à 1 ;

– l'adresse reçue est transférée dans SSPBUF ;

– le bit ACK est envoyé vers le maître sur le 9e bit d'horloge et la connexion SCL est maintenue à 0 ;

– la donnée à transmettre est chargée dans le buffer ce qui provoque le chargement du registre à décalage ;

– la connexion SCL doit être activée par la mise à 1 du bit CKP du registre SSPCON ;

– l'horloge est contrôlée par le maître, les 8 bits du mot sont sortis vers le maître sur le front descendant de l'horloge, ceci permet au signal SDA d'être valide pendant l'état haut de l'horloge ;

– une interruption peut être demandée par le passage à 1 de SSPIF au 9ème front descendant de l'horloge ;

– le bit SSPIF doit être remis à 0 par programme, SSPSTAT donne des informations sur l'état du transfert.

L'accusé de réception du maître récepteur est latché sur le front montant de la 9ème impulsion d'horloge. A ce moment :

– si SDA est à 1 (équivalent à un STOP) le transfert est terminé, l'esclave attend alors le prochain bit de START ;

– si l'ACK est valide (niveau zéro), la donnée à transmettre doit être chargée dans SSPBUF donc dans le registre à décalage et la ligne d'horloge doit être activée par CKP pour continuer le transfert.

14.4. Le fonctionnement en maître

En mode maître les lignes SDA et SCL sont pilotées par le module.
Ce mode est mis en service par les bits SSPM3 à SSPM0 et par la mise à 1 du bit SSPEN.

Dans ce cas l'utilisateur a six possibilités d'usage de la liaison définies par les bits du registre SSPCON2 :

– émettre un bit de START avec SDA et SCL par le bit SEN ;

– ou émettre un bit de START répété avec SDA et SCL par le bit RSEN ;

– écrire dans le buffer pour réaliser une transmission ;

– générer un bit de STOP avec SDA et SCL par le bit PEN ;

– configurer le port I2C pour recevoir une donnée par le bit RCEN (maître recepteur) ;

– générer un accusé de réception à la fin d'une réception avec le bit ACKEN et le bit ACKDT.

14.4.1. *Généralités*

C'est toujours le maître qui génère les impulsions d'horloge et les bits de START et de STOP.

Un transfert de données se termine par un bit de STOP ou par une répétition du bit de START.

En transmission le premier octet transmis est l'adresse de l'esclave avec qui communiquer et qui contient le bit R/W indiquant le sens de transfert de la liaison ; dans ce cas ce bit R/W est à 0 (écriture). Après cet envoi, le maître doit recevoir un accusé de réception de l'esclave puis émettre la deuxième partie de l'adresse sur 10 bits ou les données.

En réception, le premier octet transmis contient l'adresse de l'esclave avec le bit de R/W à 1 (lecture) et éventuellement la deuxième partie de l'adresse. Ensuite le maître se met en réception, reçoit les données en générant après chaque octet un accusé de réception.

L'envoi d'une donnée ou d'une adresse se fait simplement en écrivant dans le buffer, ce qui fait passer BF à 1 et met en service le décomptage du générateur de bauds afin de générer les durées TBRG.

Les bits sont décalés dans le registre à décalage vers SDA sur le front descendant de SCL. La ligne de donnée doit rester stable pendant la durée de l'impulsion d'horloge et quelque temps après le front descendant de celle-ci. Le bit de donnée doit être valide avant la montée du signal d'horloge SCL, le rester pendant la durée TBRG et aussi un peu après la descente de SCL.

Après le 8e front descendant de l'horloge, le bit BF est remis à 0 et la ligne SDA est relâchée au niveau haut en attendant l'accusé de réception ACK en provenance de l'esclave adressé (pendant le 9e bit). La valeur de l'ACK peut être déterminée par la lecture du bit ACKSTAT du registre SSPCON2.

Le maître reçoit l'ACK, le bit ACKSTAT est mis à 0 si la ligne SDA est à 0 au moment de l'échantillonnage, dans le cas contraire il est mis à 1, ce qui peut signifier que l'esclave n'a pas répondu.

Après le 9e bit, SSPIF est mis à 1, le générateur de baud est arrêté jusqu'au prochain chargement du buffer laissant SCL à l'état bas et SDA inchangé.

WCOL est mis à un quand l'utilisateur écrit dans le buffer alors qu'un transfert en série est en cours, le contenu du buffer est inchangé (l'écriture n'a pas lieu). Le programme doit prévoir la remise à 0 du bit WCOL.

14.4.2. *L'horloge*

Un générateur de baud permet le cadencement de l'horloge du maître, avec des fréquences de 100 KHz ou 400 KHz. Cette vitesse est déterminée par le contenu du registre SSPADD. Ce générateur de bauds est mis en service automatiquement par l'écriture dans le registre SSPBUF. Quand l'opération est terminée, c'est-à-dire l'accusé de réception ACK généré, le générateur s'arrête.

14.4.3. *Transmission par le maître*

– mettre à 1 le bit SEN (*Start ENable* dans le registre SSPCON2) ;

– le bit SSPIF passe à 1, le module attend le temps nécessaire pour la prise en considération du START, SEN est remis à 0 automatiquement mais SSPIF doit être remis à 0 par programme avant le retour d'interruption (si elle est autorisée) ;

– charger l'adresse à transmettre dans SSPBUF, BF passe à 1;

– l'adresse est transmise et BF repasse à 0 ;

– l'ACK de l'esclave est reçu et chargé dans le bit ACKSTAT du registre SSPCON2 ;

– une interruption peut être générée à la fin du 9e bit d'horloge par passage à 1 de SSPIF si elle est autorisée, ce bit doit être remis à 0 ;

– charger la donnée à transmettre dans SSPBUF ou la deuxième partie de l'adresse, BF passe à 1;

– la donnée est transmise en série, BF est remis à 0 ;

– le maître reçoit l'ACK de l'esclave, ACKSTAT est à 0 si l'accusé de réception reçu est conforme sinon il est mis à 1 ;

– une interruption peut être générée par SSPIF qui passe à 1 et doit être remis à 0 par logiciel ;

– le maître doit générer un bit de STOP par la mise à 1 du bit PEN du registre SSPCON2 à la fin d'une transmission qui peut comprendre plusieurs octets, afin le libérer le bus ;

– une interruption peut être générée par SSPIF dès que le bit de STOP est transmis.

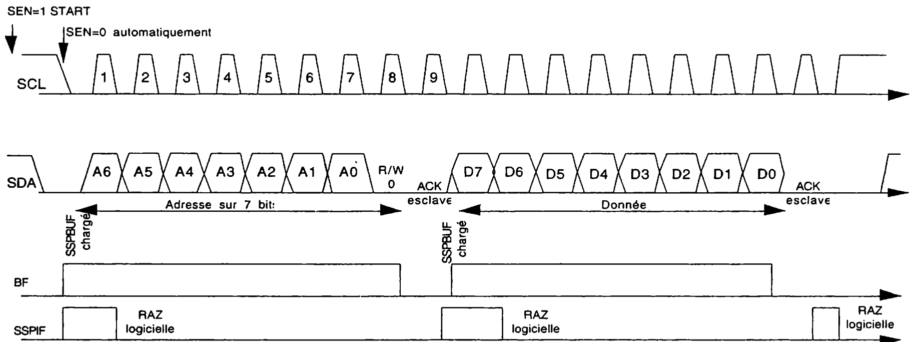


Figure 14.4. *Chronogrammes de transmission maître*

14.4.4. *Mise en réception du maître*

Le mode réception du maître est mis en service par le bit RCEN du registre SSPCON2. Le maître génère les impulsions d'horloge qui permettent à l'esclave de transmettre la donnée sur la ligne SDA.

Après le front descendant du 8e bit d'horloge, le bit RCEN est automatiquement mis à 0, le contenu du registre à décalage est transféré dans le buffer, le bit BF est mis à 1, SSPIF est mis à 1, le générateur de bauds est arrêté et la ligne SCL reste à 0.

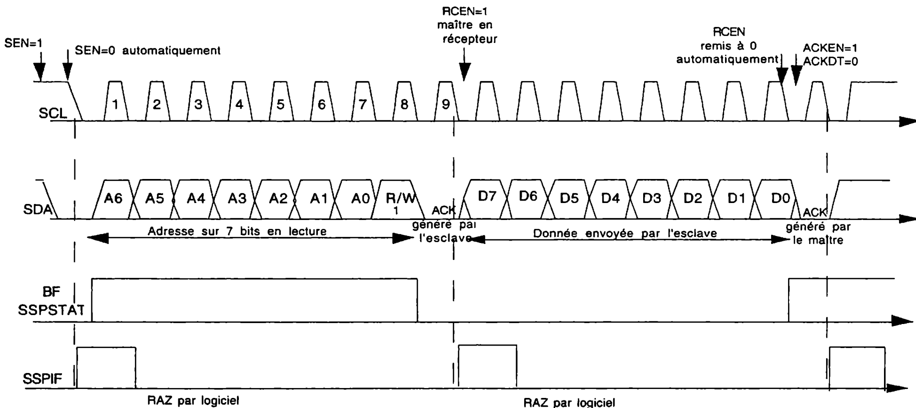


Figure 14.5. *Chronogrammes de réception du maître*

BF est remis automatiquement à 0 quand le buffer est lu.

L'utilisateur doit envoyer un ACK par la mise à 1 du bit ACKEN du registre SSPCON2.

SSPOV est mis à 1 quand 8 bits sont reçus dans le registre à décalage et que BF est déjà mis à 1 par une autre réception.

14.5. Génération du bit de START

L'utilisateur met SEN à 1.

Les deux lignes SDA et SCL sont à 1 (état de repos), le générateur de baud est chargé avec le contenu de SSPAD et le comptage démarre pour réaliser la durée du bit.

La séquence de START consiste en la mise à 0 de la ligne SDA alors que la ligne SCL est encore à 1, ce qui provoque la mise à 1 du bit S du registre SSPSTAT, ensuite la ligne SCL est mise à 0. Ensuite le générateur de Bauds recommence à compter. Dès que le comptage est terminé le bit SEN est remis à 0 automatiquement et la ligne SDA est maintenue à 0. Le bit SSPIF du registre PIR1 passe à 1.

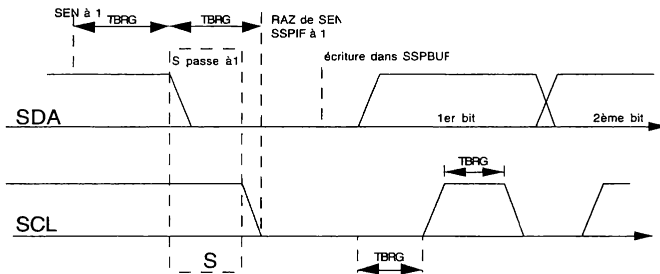


Figure 14.6. *Chronogrammes du signal START*

REMARQUE.– Si l'utilisateur charge le buffer pendant un bit de START, le bit WCOL (collision ou écrasement) du registre SSPTAT est mis à 1 et l'écriture est annulée.

14.5.1. *Génération d'un START répété*

Il permet de redémarrer une séquence de transfert sans avoir auparavant libéré le bus par un bit de STOP.

Ce type de START est déclenché par la mise à 1 du bit REN du registre SSPCON, quelque soit l'état de la ligne SDA, donc si l'opération n'est pas précédée

d'un bit de STOP. Il met SCL à 0, quand la ligne SCL est détectée à 0, le générateur de bauds est chargé et commence à décompter jusqu'à 0 (TBRG), la ligne SDA est relâchée au niveau haut. Si après cette durée BRG la ligne SDA est à 1, la ligne SCL est relachée.

Le générateur est rechargé puis décompte. Après cette durée, les lignes SDA et SCL doivent être échantillonnées à 1 et le cycle de START est généré. Le bit REN est alors mis à 0 automatiquement après la durée BRG.

Le bit SSPIF passe alors à 1, l'utilisateur peut alors envoyer le mot d'adresse.

14.6. Séquence d'*acknowledge*

Mise en service par la mise à 1 du bit ACKEN du registre SSPCON2. Dans ce cas, SCL est mis à 0, le contenu du bit ACKDT est placé sur la ligne SDA, le générateur de bauds est mis en service pour une période et SCL passe à 1. Après le temps TBRG, SCL est remis à 0, le bit ACKEN est remis automatiquement à 0.

14.7. Séquence STOP

14.7.1. *Définition*

La séquence de STOP permet de libérer le bus I2C après une transmission donc de remettre les deux lignes au repos au niveau haut.

Figure 14.7. *Chronogrammes du signal STOP*

14.7.2. *Mise en service*

Le STOP est généré par la mise à 1 du bit PEN du registre SSPCON2, dans ce cas SCL passe à 0, le générateur de bauds décompte, quand le temps TBRG est terminé, SCL passe à 1, et après une autre durée BRG la ligne SDA passe à 1.

Quand la ligne SDA est échantillonnée à 1, le bit P du registre SSPSTAT est mis à 1, TBRG plus tard le bit PEN est remis à 0 et le SSPIF passe à 1.

Nota : quand le système veut prendre le contrôle du bus I2C, il doit vérifier si le bus est occupé en lisant les bits S et P du registre SSPSTAT.

14.8. Application

Pour valider une liaison I2C, on se propose de mettre en œuvre un composant PCF8574. Celui-ci permet d'établir une liaison parallèle en lecture ou en écriture.

On étudie le logiciel minimum qui permet au P16F877 d'écrire et de lire ce port parallèle, sans toutefois faire intervenir les nombreuses possibilités de la liaison I2C.

L'adresse du PCF8574 est donnée par une adresse fixe déterminée par le composant (40h) et par la mise en place de niveaux logiques sur les connexions d'adresses du boîtier (A0-A1-A2). Celles-ci permettent de distinguer plusieurs boîtiers de même nom dans un réseau. Dans notre exemple on fixe à 44h l'adresse absolue (sans tenir compte du bit R/W) du composant (A0 = 0, A1 = 1 et A2 = 0).

Le composant PCF8574 dispose d'une adresse sur 7 bits et d'une vitesse maximum de fonctionnement de 100KHz.

Sur les quatre bits de poids forts du port parallèle on place une roue codeuse et sur les quatre bits de poids faibles des leds. Le travail consite donc à lire le port du PCF8574 et à recopier le quartet de poids fort (roue codeuse) sur le quartet de poids faible (leds). Une led branchée sur le port B indique la fin de l'opération.

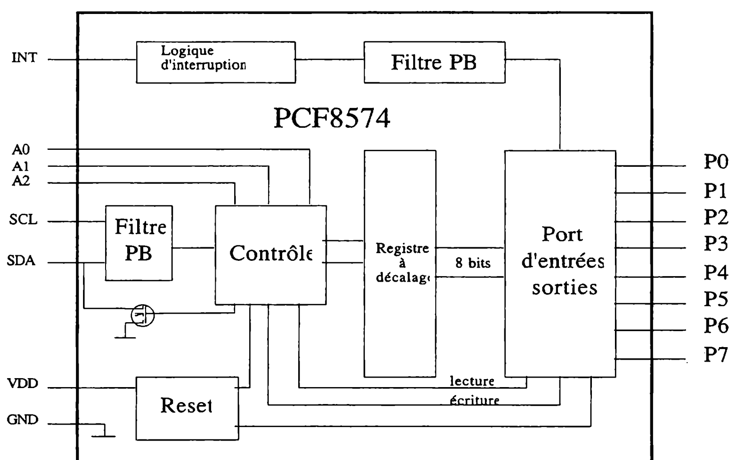


Figure 14.8. *Schéma fonctionnel du PCF8574*

Etude des registres

Le port B est en sortie donc TRISB = 0.

Les lignes RC3 et RC4 du port C, qui sont aussi les lignes SCL et SDA de la liaison I2C, sont en entrées donc TRISC = 18h.

Pour mettre en service l'interface en maître il faut que les bits SSPEN et SSPM3 du registre SSPCON soient à 1 ; donc SSPCON = 28h.

Pour générer les différentes phases opératives de la liaison il faut mettre à 1 les bits SEN, RSEN, PEN, ACKEN et RCEN du registre SSPCON2. En réception il faudra mettre un 0 dans ACKDT pour générer un accusé de réception valide, et en émission il faudra lire le bit ACKSTAT pour connaître la valeur de l'accusé de réception.

Pour connaître l'état des différentes phases opératoires il faudra tester le bit SSPIF du registre PIR1, puis le remettre à 0 avant de passer à la suivante.

Exemple de solution

```
;    PROGRAMME DE lecture/écriture dans un PCF8574     *

     LIST       p=16F877       ; Définition du processeur
     #include <P16F877.inc>    ; Définitions de variables

     __CONFIG    _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC

ADRESSE equ 0x44               ; adresse absolue du 8574 (R/W=0)
#DEFINE FOSC 4000000           ; Fréquence du quartz
#DEFINE I2CHORL 100000         ; valeur horloge transmission
#DEFINE HORLOGE (((FOSC/I2CHORL)/4)-1) ; BAUD
#DEFINE OK PORTB,1             ; indicateur de fin de travail

     CBLOCK 0x020              ; début de la zone variables
     octetlu : 1               ;
     AdresTemp : 1             ; adresse en memoire
     endc

     org 0x000                 ; Adresse de départ
     nop
     goto    init

;initialisation des registres
init bsf STATUS,RP0            ; bloc 1
     movlw b'00011000'         ; lignes RC3 et RC4 en entrées
     movwf TRISC
     clrf TRISB                ; portB en sorties
     movlw HORLOGE
```

```
     movwf SSPADD           ; choix durée BAUD
     bcf STATUS,RP0         ; bloc 0
     clrf PORTB
     movlw b'00101000'      ; SDA et SCL, clock en service
     movwf SSPCON
     bcf STATUS,RP0         ; bloc 0
     movlw ADRESSE
     movwf AdresTemp
     goto debut             ; sauter au programme principal

;envoi du bit de START
START  bsf STATUS,RP0       ; bloc 1
     bsf SSPCON2,SEN        ; envoi du start
     bcf STATUS,RP0
     call Fintache
     return
_______________________
;Lecture de l'octet
_______________________
;envoi adresse en lecture
debut    call START
     bcf STATUS,RP0
     bsf AdresTemp,0        ; met adresse en lecture
     movfw AdresTemp        ; sauvegarde temporaire
     movwf SSPBUF           ; envoi adresse
     call Fintache          ; attente SSPIF
     call recack            ; réponse esclave
     goto recept            ; vers réception

; attente fin de tâche interne
Fintache bcf STATUS,RP0
     btfss PIR1,SSPIF       ; tâche terminée?
     goto $-1
     bcf PIR1,SSPIF         ; RAZ de SSPIF
     return

; attente ACK esclave
recack    bsf STATUS,RP0    ; bloc 1
     btfsc SSPCON2,ACKSTAT  ; ack reçu = ACKSTST=0?
     goto recack            ; si oui continu
     bcf STATUS,RP0         ; bloc 0
     return

stop bsf STATUS,RP0         ; bloc 1
     bsf SSPCON2,PEN        ; envoi d'un stop
     bcf STATUS,RP0         ; bloc 0
     call Fintache
```

```
fin  goto fin

;Reception d'un octet

recept bsf STATUS,RP0          ; bloc 1
     bsf SSPCON2,RCEN          ; reception en service
     call Fintache
     bcf STATUS,RP0            ; bloc 0
     movf SSPBUF,w             ; oui lecture du buffer
     movwf octetlu             ; sauve l'octet reçu

; envoi de l'ACK maitre
     bsf STATUS,RP0            ; bloc 1
     bcf SSPCON2,ACKDT         ; init bit ACK à 0
     bsf SSPCON2,ACKEN         ; envoi ACK
     call Fintache
------------------------
; Ècriture de l'octet
------------------------
;envoi du start répété
     bsf STATUS,RP0            ; bloc 1
     bsf SSPCON2,RSEN          ; start répété
     call Fintache
     bcf STATUS,RP0            ; bloc 0

; envoi adresse en écriture
adecrit   movlw ADRESSE        ; adresse avec RW=0
     movwf SSPBUF              ; envoi adresse écriture
     call Fintache             ; attente
     call recack               ; réponse esclave ?
     goto donecrit             ; si ACK reçu envoi donnée

; envoi d'une donnée
donecrit
     swapf octetlu,w           ; inversion des quartets
     movwf SSPBUF              ; charge buffer pour l'envoi
     call Fintache             ; attente
     call recack               ; reponse esclave ?
     bsf OK                    ; indicateur fin de programme
     call stop                 ; envoi d'un stop

fin  goto fin                  ; boucle sans fin
     END
```

Chapitre 15
Débuter avec MPLAB

MPLAB est le logiciel de développement proposé par la société MICROCHIP pour le développement des microcontrôleurs PIC.

Au lancement de MPLAB apparaît une fenêtre ; la barre du haut propose des menus déroulants et contient les icônes de la barre d'outils.

La barre du bas est une barre d'état qui renseigne le programmeur sur le fonctionnement de l'étude.

La signification des icônes de la barre d'outils est donnée à la place de la barre d'état au moment du survol.

MPLAB IDE - C:\PIC\ESSAI1.PJT
File Project Edit Debug PICSTART Plus Options Tools Window Help
ROM RAM SFR
1 2 3 4 | 1 1 3 4 | 1 2 3 4 5 | 1 2 3 4 5

1 - changer barre d'outil
2 - ouvrir un projet
3 - Sauvegarder projet
4 - rechercher un texte

1 - couper
2 - coller dans calepin
3 - coller au curseur
4 - sauvegarder fichier

1 - mise en pas à pas
2 - arrêt pas à pas
3 - pas à pas normal
4 - pas à pas hors boucle
5 - reset du PIC

1 - visualiser la ROM
2 - visualiser la RAM
3 - voir registres spéciaux
4 - voir d'autres variables
5 - assembler le projet

Ln 1 Col 2 | 32 | # | WR | No Wrap | INS | PIC16F84 | pc:0x00 | w:0x00 | - - z dc c | Bk On | Sim | 4 MHz
1 2 3 4 5 6 | 1 2 3 | 1 2 3 4

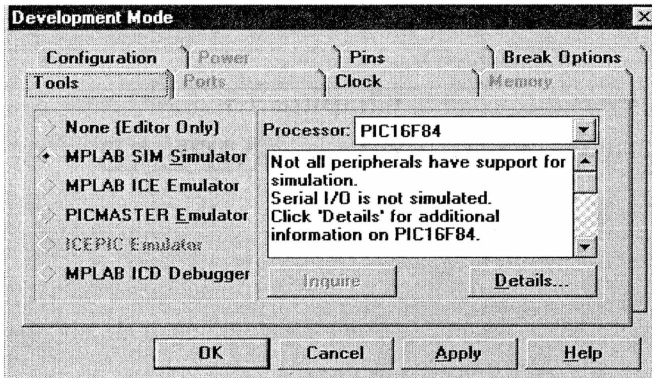
1 - Posistion du curseur
2 - nombre de lignes du fichier`
3 - fichier modifié
4 - fichier lecture/écriture
5 - pas de retour ligne automatique
6 - mode insertion de caractère

1 - type de micro-contrôleur
2 - contenu du PC
3 - contenu de W

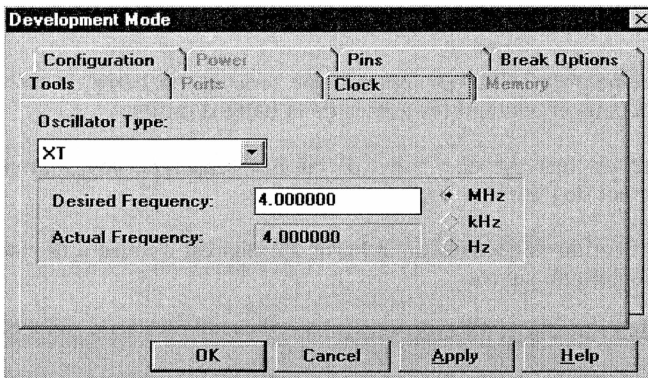
1 - contenu du registre d'état
2 - points d'arrêt autorisés
3 - mode simulateur
4 - quartz 4 MHz

15.1. Déterminer la configuration de développement

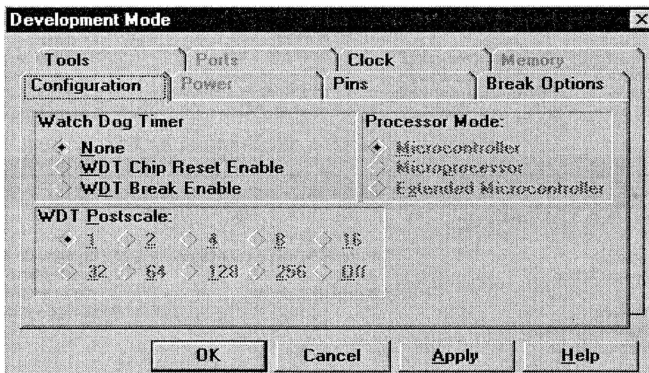
– Choisir les outils (*Tools*) Processor et MPLAB SIM pour le simulateur :



– Choisir le type d'horloge et sa fréquence, Clock :



– Choisir la configuration du chien de garde et du microcontrôleur.



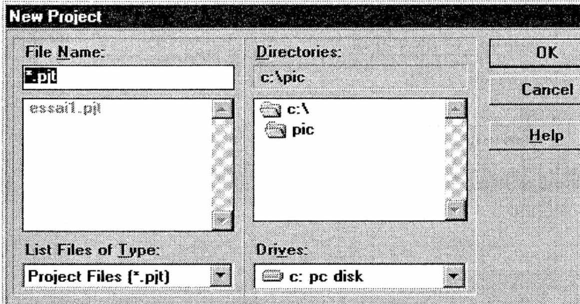
15.2. Mettre en place le projet

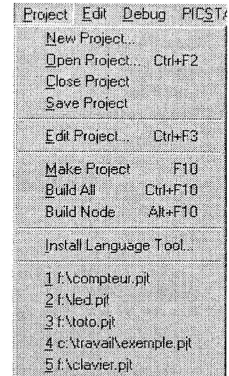
Un projet est un fichier qui contient toutes informations nécessaires à son étude telles que les noms des fichiers liés etc; son extension imposée est « .pjt ».

Dans le menu PROJECT sélectionner NEW PROJECT

Choisir l'emplacement de travail : exemple un dossier C:\PIC

Donner un nom dans la case FILENAME puis OK.





Une fenêtre EDIT PROJECT s'ouvre :

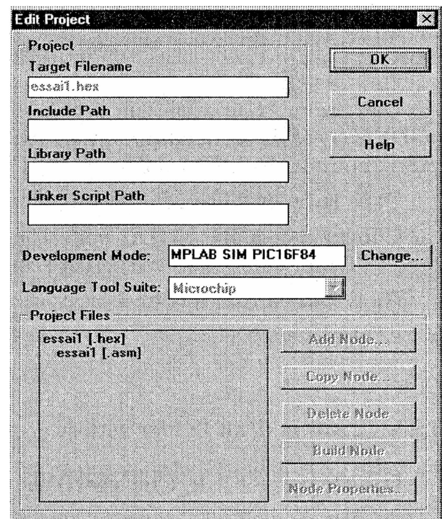
Première case on trouve le nom du fichier.hex, c'est le nom du fichier cible donné au projet avec l'extension « .hex ».

Deuxième case : INCLUDE PATH chemin des fichiers « include.inc » à utiliser s'ils ne sont pas dans le dossier de travail.

Troisième case : LIBRARY chemin des librairies de sous-programmes que l'on veut utiliser.

Quatrième case : LINKER pour utiliser l'éditeur de liens

Dans la fenêtre *Developpement mode* : cette fenêtre est aussi accessible à partir du menu OPTION de la fenêtre principale.

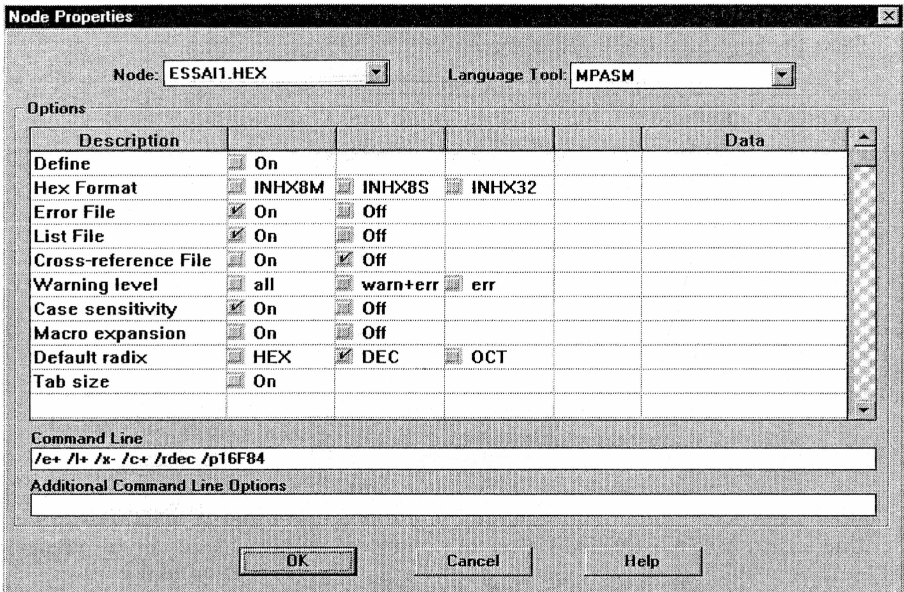


Clic sur CHANGE :

- on choisit le microcontrôleur utilisé ;
- on choisit de travailler avec le simulateur : MPLAB-SIM ;
- le langage utilisé est celui de Microchip (MPASM) ;
- OK.

Le projet est créé.

Après avoir sélectionné le nom du fichier **.hex** de la grande fenêtre de Edit project, choisir la case NODE PROPERTIES qui s'est noircie.



Dans cette boîte, dialogue cochez les options, notamment **DEFAULT RADIX =>DEC** (type de numération) qui permet à l'assembleur de prendre les nombres sans préfix comme décimaux, puis clic sur OK.

Dans la fenêtre Project Files apparaît le fichier par défaut essai1.hex.

Cliquer sur **ADD NODE** pour ajouter un nœud (il s'agit de lier un fichier au projet), l'explorateur s'ouvre on peut écrire un nom de fichier : essai1.asm, puis OK.

Au retour dans la fenêtre Edit project cliquez sur OK.

Le projet est prêt.

15.3. Création d'un fichier source

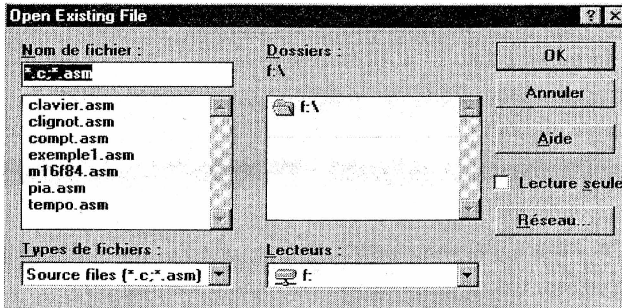
Dans le menu FILE, Clic sur NEW.

Une fenêtre vide s'ouvre, on peut y écrire alors le source du programme à l'étude.

FILE => SAVE AS et sauver le fichier vide comme TOTO.ASM

Clic sur la fenêtre vide, rédiger le source assembleur puis le sauvegarder.

On peut aussi ouvrir un fichier déjà existant dans la liste déroulante.



15.4. Assemblage du programme

```
f:\tempo.asm
 list p=16F84
 include "p16F84.inc"

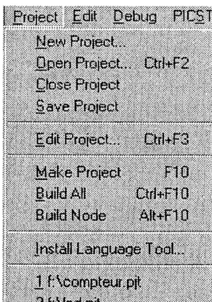
;declarations des variables
        CBLOCK 0x0C
        T1 : 1
        T2 : 1
        T3 : 1
        ENDC

;équivalences => permettent de modifier
;la durée de la tempo
d1 equ 6
d2 equ 217

        org 0x00
        goto tempo

;sous-programme de temporisation
tempo   clrf T2         ; met T2 a 0
        movlw d1                ; charge 6 dans W
        movwf T3        ; place 6 (W) dans le registre T3
        movlw d2        ; charge 217 dans W
        movwf T1        ; met 217 dans T1
BOU     decfsz T1,f     ; decremente T1, si resultat = 0
        goto BOU        ; branche a BOU
        decfsz T2,f     ; decremente et test
        goto BOU
        decfsz T3,f     ; decremente et test
        goto BOU
```

Exemple de source



L'assembleur va assembler (ou compiler) le fichier source du projet sans tenir compte de ce qui visible à l'écran et va créer un fichier .hex de même nom.

Pour lancer l'assemblage CTRL-F10 ou encore : PROJECT => BUILD ALL.

Une fenêtre bilan (*build Results*) s'ouvre dans laquelle les avertissements (*warnings*) et les erreurs sont signalés.

Les avertissements et les messages n'empêchent pas l'assemblage et ne font qu'attirer l'attention du programmeur sur des ambiguités.

On peut retrouver la ligne où se trouve l'erreur dans le source avec la commande GO TO LINE du menu Edit.

La corriger après avoir cliqué sur la fenêtre contenant le source.

Dans l'exemple ci-dessous, il y a deux erreurs ; une à la ligne 1 et une à la ligne 23.

```
Build Results
Building TEMPO.HEX...

Compiling TEMPO.ASM:
Command line: "C:\PROGRA~1\MPLAB\MPASMWIN.EXE /e+ /l+ /x- /c+ /rdec /p16F84 /q F:\TEM
Error[108]   F:\TEMPO.ASM 1 : Illegal character (4)
Error[113]   F:\TEMPO.ASM 23 : Symbol not previously defined (d3)

MPLAB is unable to find output file "TEMPO.HEX". This may be due to a compile, assemb

Build failed.
```

Recommencer l'assemblage (F10) jusqu'à ce que le message BUILD COMPLETED SUCCESSFULLY apparaisse, dans ce cas le fichier .hex est utilisable et peut être programmé dans la mémoire du microcontrôleur.

15.5. Simulation du programme

Le programme peut être essayé avec le simulateur en exécution automatique, automatique ralentie, en mode pas à pas ou en mode pas à pas hors boucle (*step over*).

15.5.1. *Définir les informations à surveiller*

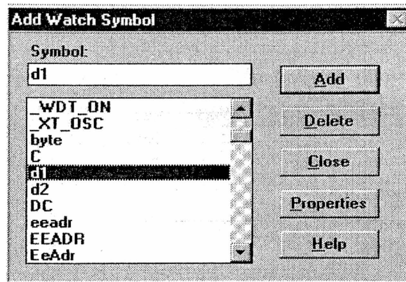
Pendant la simulation on peut visualiser sur l'écran un grand nombre de variables : les registres, les registres spéciaux, les ports et aussi des variables que l'on a choisies dans le programme en cours de mise au point.

WINDOW => SPECIAL FONCTION REGISTERS pour visualiser le contenu des registres en binaire, hexadécimal et en décimal. Les valeurs visualisées évoluent au fur et à mesure du déroulement du programme.

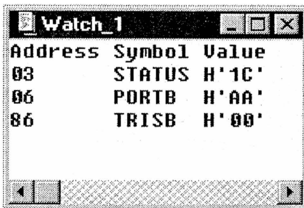
Special Function Register Window

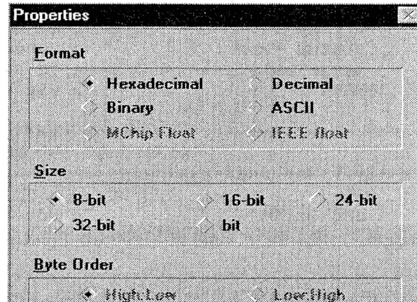
SFR Name	Hex	Dec	Binary	Char
tmr0	00	0	00000000	.
pcl	00	0	00000000	.
option_reg	FF	255	11111111	.
status	18	24	00011000	.
fsr	00	0	00000000	.
porta	00	0	00000000	.
trisa	1F	31	00011111	.
portb	00	0	00000000	.
trisb	FF	255	11111111	.
eedata	00	0	00000000	.
eecon1	00	0	00000000	.
eeadr	00	0	00000000	.
eecon2	00	0	00000000	.
pclath	00	0	00000000	.
intcon	00	0	00000000	.
w	00	0	00000000	.
t0pre	00	0	00000000	.

WINDOW => WATCH WINDOWS => NEWWATCH WINDOWS dans la fenêtre qui s'ouvre ; choisir la variable à visualiser.



Puis clic sur **PROPERTIES** et choisir le format d'affichage de la donnée. **ADD WATCH SYMBOL** fermer.





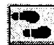
On peut alors organiser l'écran avec toutes les fenêtres ouvertes, l'une contenant le source et sur laquelle on verra évoluer par une barre horizontale le déroulement du programme et les autres visualisant toutes les variables intéressantes.

15.5.2. *La simulation*

Une fois toutes les variables à surveiller visibles sur l'écran on peut commencer la simulation :

DEBUG => RUN => RESET ou encore F6, le curseur vient se mettre sur la première ligne du programme dans la fenêtre du source.

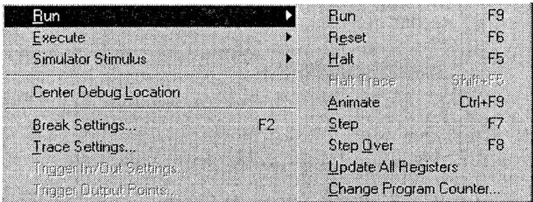
Dans la barre d'état on remarque que le PC est chargé avec 0x00.

15.5.2.1. *Pas à pas normal (*STEP *ou F7)* 

En pressant sur F7 ou en cliquant sur l'icône du pas à pas, on exécute du pas à pas approfondi (le pas à pas s'effectue dans les boucles et les sous-programmes), à

chaque appui on peut constater l'évolution des registres et des variables. Une ligne surlignée permet de suivre l'évolution dans le source.

Un appui sur la touche F5 ou encore sur l'icône au feu rouge, permet d'arrêter le mode pas à pas.




```
c:\pic\essai1.asm
;exemples
;---------
        CBLOCK 0x00C              ; début de la zone variables
        w_temp :1                 ; Zone de 1 byte
        status_temp : 1           ; zone de 1 byte
        mavariable : 1            ; je déclare ma variable
       ENDC                       ; Fin de la zone

;*********************************************************************
;                    DEMARRAGE SUR RESET
;*********************************************************************

        org 0x000                 ; Adresse de départ après reset
        goto    start             ; Adresse 0: initialiser

;*********************************************************************
;                   ROUTINE INTERRUPTION
;*********************************************************************

                      ;sauvegarder registres
                      ;---------------------
        org 0x004                 ; adresse d'interruption
```

Instruction exécutée

15.5.2.2. *Pas à pas hors boucle (*Step over *ou F8)*

Le pas à pas hors boucle (F8) n'entre pas dans les sous-programmes.

15.5.2.3. *Exécution automatique*

Un appui sur F9 lance le programme à partir du curseur, l'évolution des registres n'est plus visible tant que le programme n'est pas arrêté par la commande **HALT** (F5) du menu DEBUG

Animate ou Ctrl F9 fait avancer le programme plus lentement. L'évolution des registres reste visible.

15.6. Les points d'arrêt

Le programme s'arrête quand il rencontre un point d'arrêt, on peut alors vérifier le résultat des instructions en lisant les contenus des registres.

Pour mettre en place un point d'arrêt : clic bouton droit à l'endroit ou l'on veut s'arrêter ; dans le menu qui apparaît choisir **BREAK POINTS**.

Le curseur marqué change de couleur.

On peut aussi demander au programme de s'exécuter jusque là par **RUN TO HERE** ou mettre des points d'arrêt conditionnels (**TRACE** ou **TRIGGERS**).

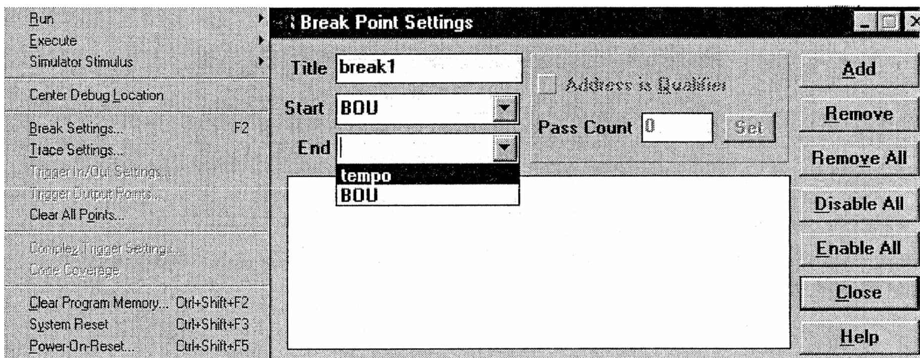
F6 puis F9 fait se dérouler le programme jusqu'à ce point.

On peut placer plusieurs points d'arrêt, pour les supprimer : choisir **CLEAR ALL POINT** dans le menu **DEBUG**.

15.7. Mise au point dans une boucle

Menu **DEBUG** => **TRACE SETTINGS**
- donner un nom (Title)
- adresses (*Start* et *End*) de la zone à tracer (étiquettes) puis Add

Sur la première ligne de la grande fenêtre clic



Adresse is qualifier => mettre un chiffre dans le compteur de passage (*Pass Count*) puis **SET.**

Fermer la enêtre (CLOSE) puis lancer le test => F6 => F9.

Enlever les points d'arrêt.

15.8. Simuler avec un événement extérieur

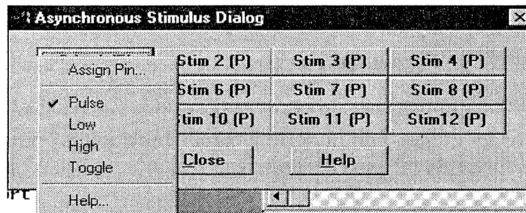
Menu DEBUG => **Simulator Stimulus** => **Asynchronous Stimulus Dialog**

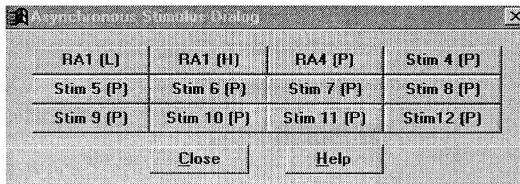
Une fenêtre avec 12 événements configurables s'ouvre.

Clic droit sur STIM 1(P) => un menu déroulant apparaît => **assign pin** permet de choisir la connexion à simuler => le bouton prend le nom de la connexion RB0(P), avec le bouton droit on peut aussi choisir le mode de fonctionnement du bouton (*pulse* : sensible à une impulsion, *low* : sensible à l'état bas, *high* : sensible à l'état haut, *toggel* : inversion à chaque pression).

Si on choisit « low » le bouton devient RB0(L). on peut en créer un autre avec le même nom mais sensible à un autre changement (voir exemple ci-dessous).

Après chaque pression sur le bouton (stimuli) il faut revenir à la fenêtre du source et faire avancer d'un pas le simulateur. Dans le cas d'une interruption le pas suivant se trouve en 0x04 début du sous-programme d'interruption.





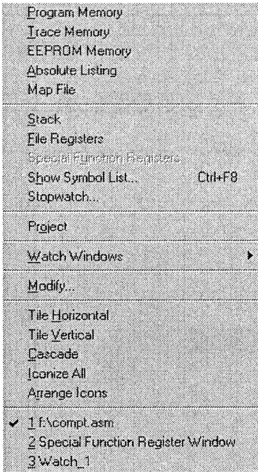
EXEMPLE.– Un clic sur RA1(L) va faire passer la connexion à l'état bas, RA1(H) à l'état haut et RA4(P) va recevoir une impulsion.

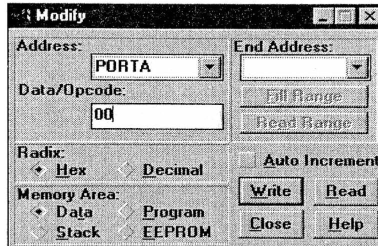
15.9. Modification des registres pendant la simulation

Menu **WINDOW** => **MODIFY**

Choisir dans la liste proposée des **Address** le registre à modifier, écrire la valeur à modifier dans la fenêtre Data/Opcode puis WRITE.

Dans l'exemple ci-dessus on écrit 00 dans le registre PORTA, il faut ensuite faire avancer d'un pas le programme pour prendre en compte cette modification.





On peut aussi modifier les données de toute une zone d'adresses dans ce cas il faut aussi remplir End Address.

ANNEXE 1
Les registres du PIC 16F877

Addresses	Nom	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	POR,BOR
16F877 - BLOC 0										
00h	INDF	Donne le contenu·l'adresse pointée par FSR addressage indirect								0000 0000
01h	TMR0	Registre Timer0								xxxx xxxx
02h	PCL	PC LSB								0000 0000
03h	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx
04h	FSR	Pointeur adressage indirect								xxxx xxxx
05h	PORTA	—	—	Latch du PORTA en écriture: PORTA en lecture						- -0x 0000
06h	PORTB	PORTB Latch en écriture : PORTB en lecture								xxxx xxxx
07h	PORTC	PORTC Latch en écriture : PORTC en lecture								xxxx xxxx
08h	PORTD	PORTD Latch en écriture : PORTD en lecture								xxxx xxxx
09h	PORTE	—	—	—	—	—	RE2	RE1	RE0	- - - - -xxx
0Ah	PCLATH	—	—	—	5 bits MSB du PC					- - -0 0000
0Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x
0Ch	PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000
0Dh	PIR2	—	réservé	—	EEIF	BCLIF	—	—	CCP2IF	-r-0 0- -0
0Eh	TMR1L	Maintien du L S B du Registre 16 bits TMR1								xxxx xxxx
0Fh	TMR1H	Maintien du M S B du Registre 16 bits TMR1								xxxx xxxx
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	- -00 0000
11h	TMR2	Registre Timer2								0000 0000
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000
13h	SSPBUF	Buffer Reception/ RegistreTransmission - Série Synchrone								xxxx xxxx
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000
15h	CCPR1L	Capture/Compare/PWM Registre1 (LSB)								xxxx xxxx
16h	CCPR1H	Capture/Compare/PWM Registre1 (MSB)								xxxx xxxx
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	- -00 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x
19h	TXREG	USART Registre transmission								0000 0000
1Ah	RCREG	USART Registre réception								0000 0000
1Bh	CCPR2L	Capture/Compare/PWM Registre2 (LSB)								xxxx xxxx
1Ch	CCPR2H	Capture/Compare/PWM Registre2 (MSB)								xxxx xxxx
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1 C	CP2M0	- -00 0000
1Eh	ADRESH	A/D Registre Résultat MSB								xxxx xxxx
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON	0000 00-0

x = inconnu, u = inchangé, - = non utilisé (lu donne 0), q = la valeur dépend des conditions, r : réservé.

Addresses	Nom	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	POR,BOR
16F877 - BLOC 1										
80h	INDF	contient la donnée pointée par FSR adressage indirect								0000 0000
81h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111
82h	PCL	PC LSB								0000 0000
83h	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx
84h	FSR	Pointeur adressage indirect								xxxx xxxx
85h	TRISA	—	—	PORTA registre de direction						- -11 1111
86h	TRISB	PORTB registre de direction								1111 1111
87h	TRISC	PORTC registre de direction								1111 1111
88h	TRISD	PORTD registre de direction								1111 1111
89h	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE reg dir			0000 -111
8Ah	PCLATH	—	—	—	5 bits MSB du PC					- - -0 0000
8Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x
8Ch	PIE1	PSPIE (2)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000
8Dh	PIE2	—	réservé	—	EEIE	BCLIE	—	—	CCP2IE	- r -0 0- -0
8Eh	PCON	—	—	—	—	—	—	POR	BOR	- - - - - -qq
8Fh	—	Non utilisé								—
90h	—	Non utilisé								—
91h	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000
92h	PR2	Registre période du Timer2								1111 1111
93h	SSPADD	Registre Addresse Port série mode I 2 C								0000 0000
94h	SSPSTAT	SMP	CKE	D/A	P	SR	/W	UA	BF	0000 0000
95h	—	Non utilisé								—
96h	—	Non utilisé								—
97h	—	Non utilisé								—
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010
99h	SPBRG	Contrôle générateur de Bauds								0000 0000
9Ah	—	Non utilisé								—
9Bh	—	Non utilisé								—
9Ch	—	Non utilisé								—
9Dh	—	Non utilisé								—
9Eh	ADRESL	A/D Resultat LSB								xxxx xxxx
9Fh	ADCON1	ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0	0- - - 0000

x = inconnu, u = inchangé, - = non utilisé (lu donne 0), q = la valeur dépend des conditions, r : réservé.

Adresses	Nom	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	BOR-POR
16F877 - BLOC 2										
100h	INDF	Contient la donnée de l'adresse pointée par FSR								0000 0000
101h	TMR0	Registre du module Timer0								xxxx xxxx
102h	PCL	PC Poids faible								0000 0000
103h	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx
104h	FSR	Pointeur d'adressage indirect								xxxx xxxx
105h	—	Inutilisé								—
106h	PORTB	PORTB Latch en écriture ; PORTB en lecture								xxxx xxxx
107h	—	Inutilisé								—
108h	—	Inutilisé								—
109h	—	Inutilisé								—
10Ah	PCLATH	—	—	—	5 bits poids fort du PC					- - -0 0000
10Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x
10Ch	EEDATA	EEPROM Registre de données LSB								xxxx xxxx
10Dh	EEADR	EEPROM Registre d'adresses LSB								xxxx xxxx
10Eh	EEDATH	—	—	EEPROM Registre de données poids fort						xxxx xxxx
10Fh	EEADRH	—	—	—	EEPROM Registre d'adresse poids fort					xxxx xxxx
16F877 - BLOC 3										
180h	INDF	Contient la donnée de l'adresse pointée par FSR								0000 0000
181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111
182h	PCL	PC poids faible								0000 0000
183h	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx
184h	FSR	Pointeur d'adressage indirect								xxxx xxxx
185h	—	Inutilisé								—
186h	TRISB	PORTB Registre de direction								1111 1111
187h	—	Inutilisé								—
188h	—	Inutilisé								—
189h	—	Inutilisé								—
18Ah	PCLATH	—	—	—	5 bits poids fort du PC					- - -0 0000
18Bh	INTCON	GIE	PEIE	T0IE INTE	RBIE	T0IF	INTF	RBIF		0000 000x
18Ch	EECON1	EEPGD	—	—	—	WRERR	WREN	WR	RD	x- - - x000
18Dh	EECON2	EEPROM Registre de contrôle 2								- - - - - - - -
18Eh	—	Reservé								0000 0000
18Fh	—	Reservé								0000 0000

x = inconnu, u = inchangé, - = non utilisé (lu donne 0), q = la valeur dépend des conditions, r : réservé.

Annexe 2

Les microcontrôleurs PIC sont une production de la société :

Microchip Technology Inc
Chandler – Arizona U.S.A.

PIC est une marque déposée de MICROCHIP.

MPLAB est un logiciel outil de développement fourni gratuitement par la société Microchip. Il comprend un éditeur, un assembleur et un simulateur. Il permet la programmation des composants et le « deboggage » en connectant entre l'ordinateur et le projet une carte d'émulation (MPLAB ICD1).

Nota : au moment de la rédaction de cet ouvrage, Microchip proposait MPLAB version 5, depuis il a développé MPLAB version 6 avec une interface graphique plus conviviale et plus détaillée et aussi la possibilité de connecter un émulateur ICD2 *via* la connexion USB de l'ordinateur.
Le lecteur n'aura aucune difficulté a retrouver les fonctionnalités exposées ci-dessus dans la version 6 de MPLAB.

Bibliographie :
Documentations :

Microchip DS 30292C
Microchip DS 30430C
et diverses notes d'application.

Le logiciel et la documentation sont disponibles sur le serveur :

www.microchip.com

CET OUVRAGE A ÉTÉ COMPOSÉ PAR HERMÈS SCIENCE PUBLICATIONS ET ACHEVÉ D'IMPRIMER PAR L'IMPRIMERIE FLOCH À MAYENNE EN DÉCEMBRE 2003.

DÉPÔT LÉGAL : DÉCEMBRE 2003.
N° D'IMPRIMEUR : 58727.
Imprimé en France

Fabriqués et commercialisés depuis de nombreuses années, les microcontrôleurs PIC sont reconnus pour leur rapidité et pour leur facilité de programmation *in situ*.

La série FLASH permet entre autres de nombreux essais et reprogrammations lors de la phase d'étude. Leur architecture permet des opérations rapides, les instructions sont peu nombreuses et leur interfaçage est complet : ports parallèles, convertisseurs analogiques-numériques, temporisateurs, et liaisons série synchrone et asynchrone. Ils sont déclinés en de très nombreuses versions et chaque application trouve un modèle adéquat.

Cet ouvrage contient une étude complète du 16F84 afin de se familiariser avec l'architecture PIC et sa programmation, une étude succincte des interfaces du 16F877, une étude plus approfondie des interfaces série synchrone (SPI et I2C) de plus en plus employées dans l'interfaçage électronique et une prise en main du simulateur du logiciel de développement MPLAB.

Les microcontrôleurs PIC s'adresse aux élèves et aux enseignants des lycées techniques et professionnels, aux étudiants des IUT et BTS génie électrique et électronique, aux élèves des grandes écoles d'ingénieurs. Il sera aussi très utile à tous les utilisateurs professionnels de ce type de mat…



L'auteur

Bernard Béghyn fait bénéficie… …érience pédagogique et technique qu'il a… …breuses années, en tant que professeu… …sse de techniciens supérieurs. On trouv… …chez le même éditeur un ouvrage équivalent sur les microcontrôleurs 68HC11.

ISBN 2-7462-0764-8



Hermes Science